FEDERAL STUDENT AID

We Help Put America Through School

EAI Implementation Workshop Part 2

14 May, 2002

1:30-4:00 PM

Room 221 BC

Dial-In: (877) 714-4281

Meeting ID: 4057

FEDERAL STUDENT AID
We Help Put America Through School

Agenda

Part I: Overview of EAI & Intro To EAI Concepts

Overview of EAI

Intro to EAI Concepts

Break

Part II: Products & Architecture

Products

Overview of the EAI Architecture

Questions & Answers

Part 2: Products & EAI Architecture

Products

- **-Overview**
- **-MQSeries**
- **-AMI**
- **-MQSI**
- **-Data Integrator**

Examples

Questions & Answers

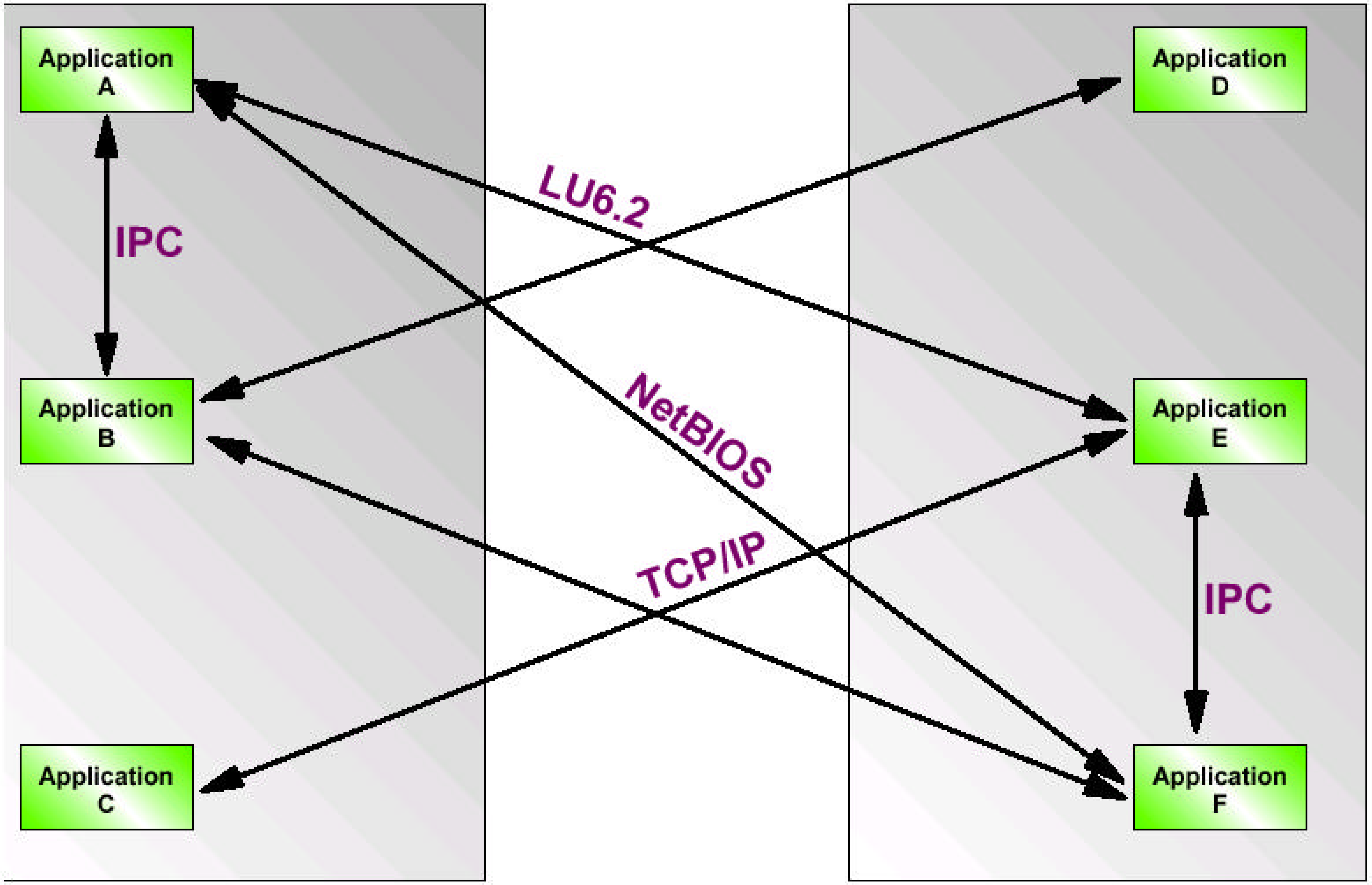
The EAI Core Team installs, configures and administers selected EAI products.

Product/Tool	Description
MQSeries	Transport of messages between systems
MQSI	Transformation and formatting of messages between systems
Data Integrator	Large amount of data transportation
Adapters	Enables program interfaces from MQSeries to application.
AMI	Simplified messaging API for business application programs.

FEDERAL STUDENT AID
We Help Put America Through School

MQ – Before

Before messaging



FEDERAL STUDENT AID
We Help Put America Through School

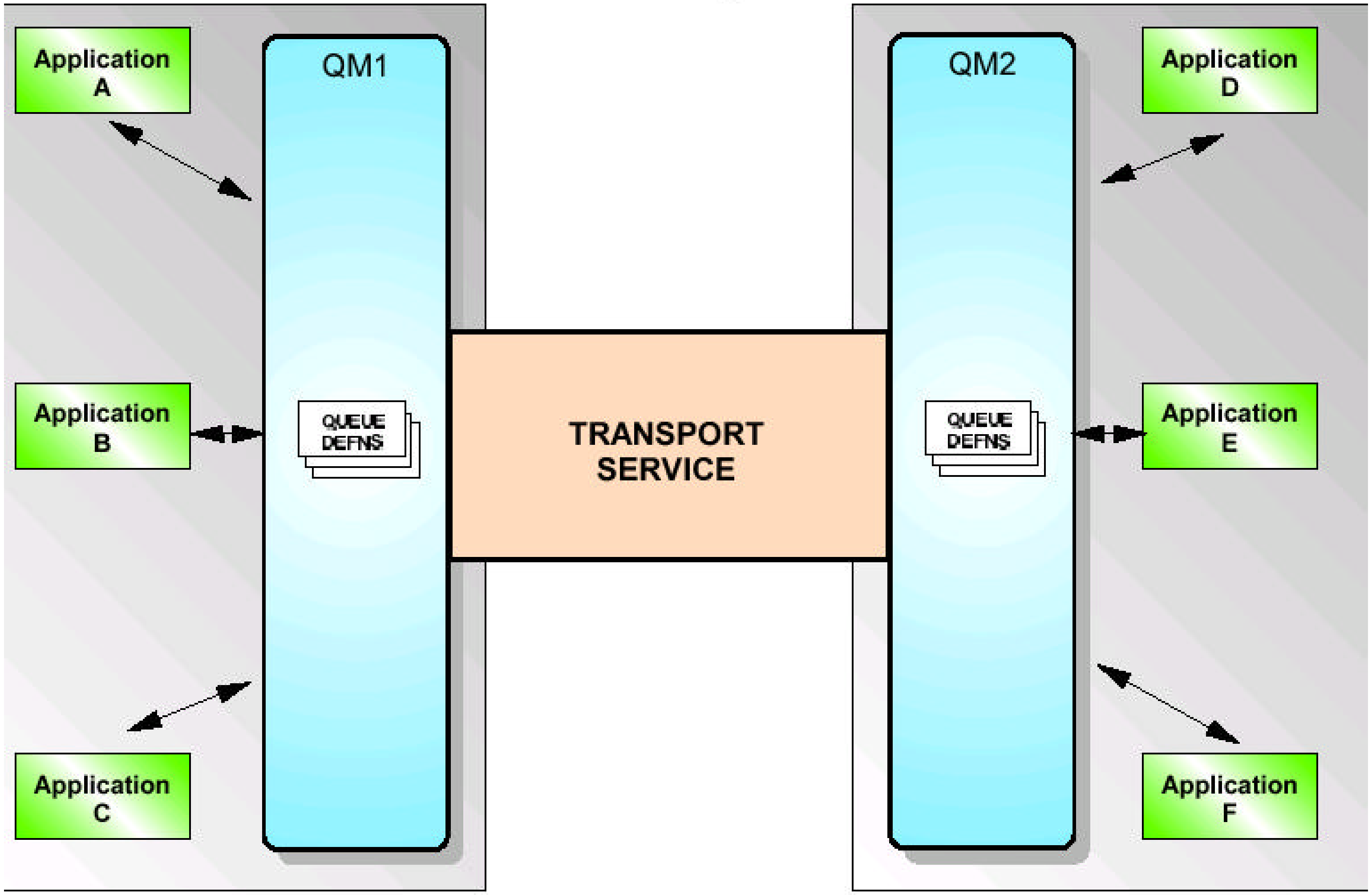
MQ-Before Messaging Notes

- The days of stand-alone business applications are pretty much over. Nowadays applications are merely a piece of the solution which must be able to communicate both upstream and downstream with partner applications. These partner applications are often written by different development organisations and are located across the enterprise or even in other enterprises. The traditional approach of writing a TCP/IP application is just too complicated in modern business architectures. An application writer must concern himself with
  - Network topology/machine addresses
  - Data representation and formats
  - Application availability and maintainance schedules
  - Complicated network programming
    - Particularly if communicating with more than one partner.
  - Guaranteed delivery
    - Writing two-phase commit protocols and handling indoubt situations
- Most, if not all, of these concerns can be removed by a messaging backbone.

FEDERAL STUDENT AID
We Help Put America Through School

MQ Messaging

With Websphere MQ



FEDERAL STUDENT AID
We Help Put America Through School

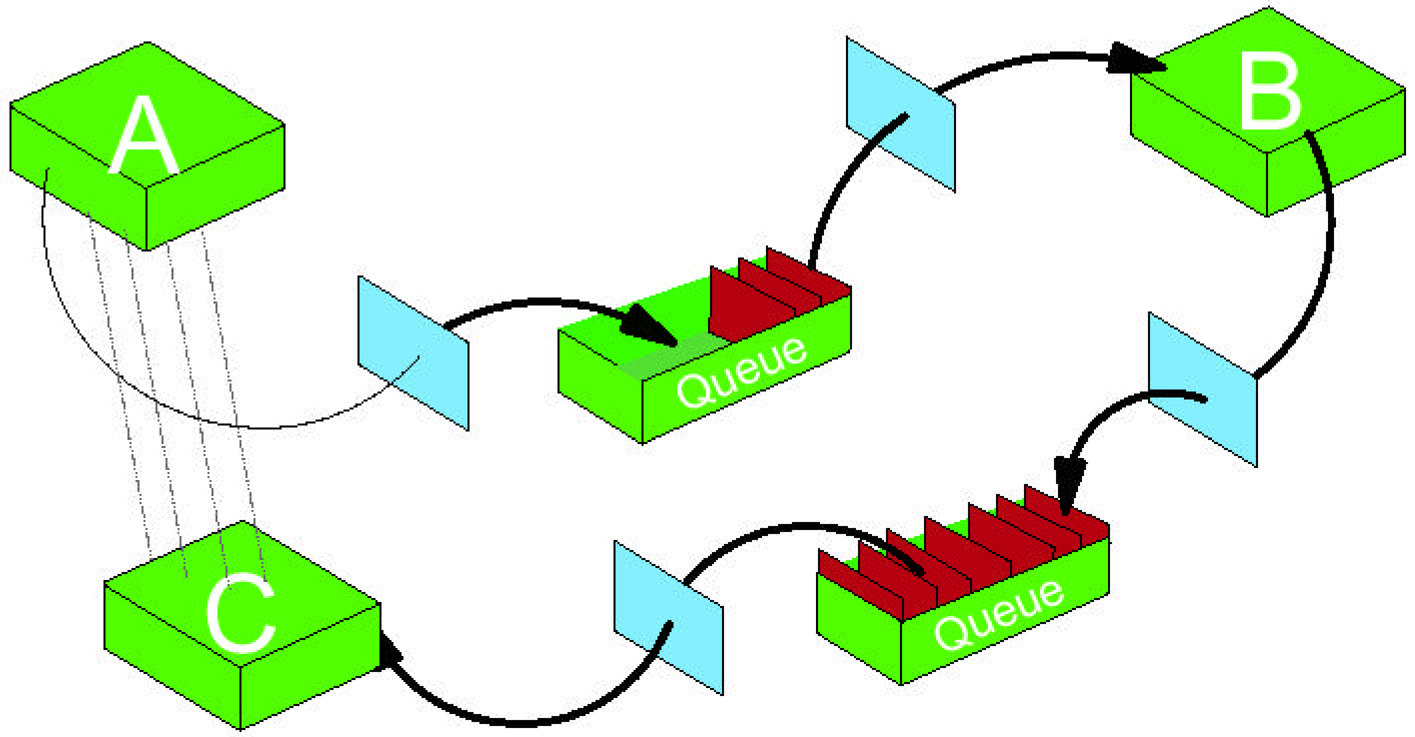
With MQ Notes

- The introduction of Websphere MQ vastly simplifies the work the application need do. An application now only needs to put a message to the correct queue and let MQSeries worry about how to deliver the message across any network. The messaging paradigm, similar to in and out trays, lends itself to the concepts of being one part in a chain of business processes.

- Application development is simplified. Often an application can be fully developed on a single machine without any communications and then deployed across the network without any changes.

- Management of the messaging network, the location of the servers etc is now part of the MQSeries configuration and not part of the application leading to simpler more reliable, flexible designs.

Overall Model



Time Independence
Location Transparency
Platform Independence
Many-to-Many Connectivity

MQ – Overall Model Notes

- MQSeries Messaging has a number of architectural goals
  - Time independence
  - Location Transparency
  - Platform Independence
  - Many-to-Many Connectivity
- The purpose of this presentation is to show how these goals are achieved when going across networks of Queue Managers.
  We can even add a couple of new goals
  - Protocol independence (TCP/IP, SNA etc)
  - Application unaware

FEDERAL STUDENT AID
We Help Put America Through School

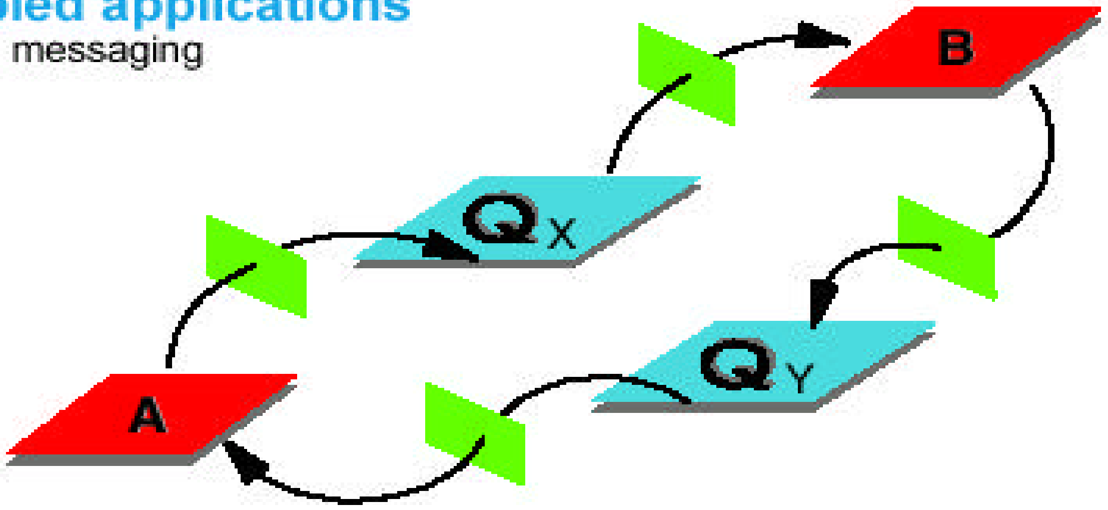
Messaging Fundamentals?

A single, multi-platform API
- Easy to use ... message centric interface
- Network independent

... faster application development

Assured message delivery

Loosely-coupled applications
- Asynchronous messaging



Reducing Complexity

Reliable, distributed computing

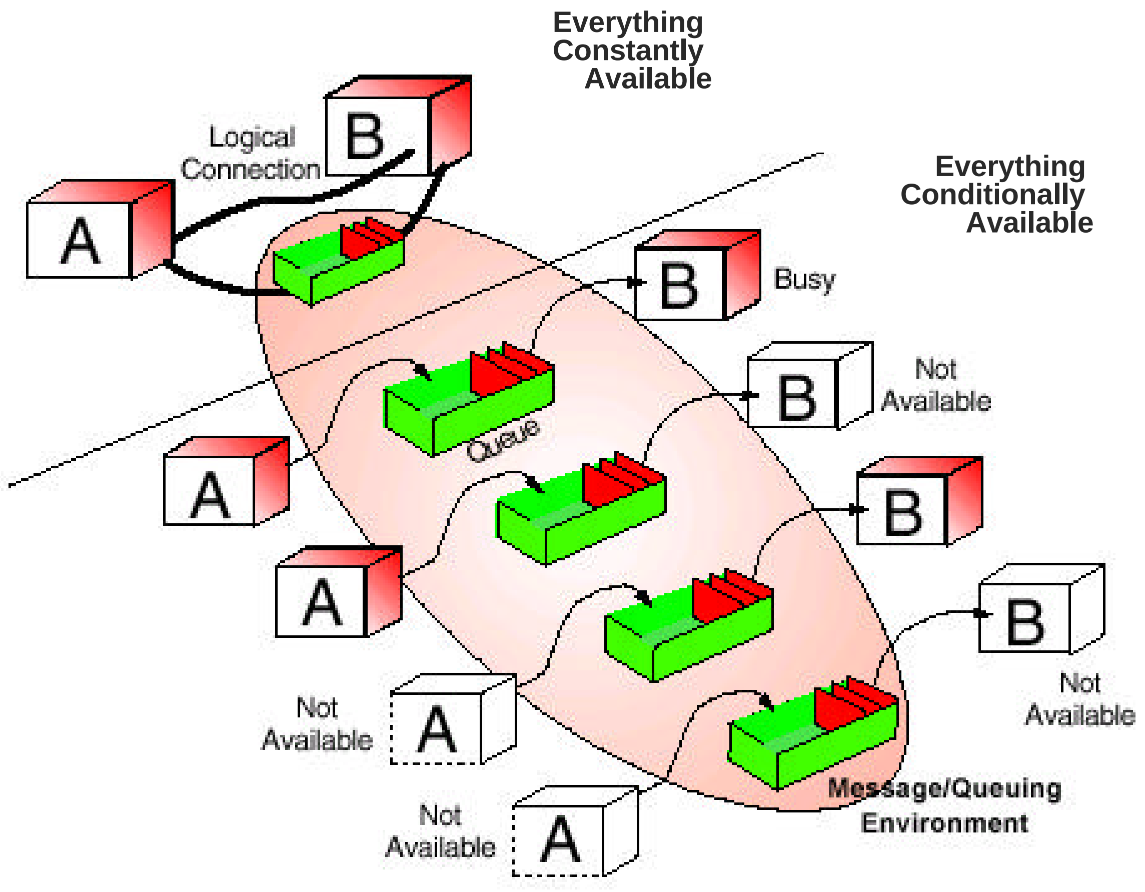
- Complex
- Error prone

MQSeries eases the complexity

MQSeries is *not* a substitute for:

- Well written applications
- Robust network
- Good operational procedures
- Well managed system

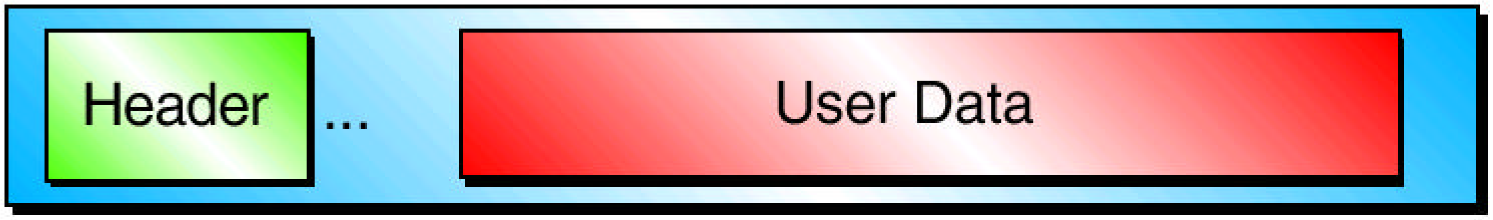
Availability Choices


Everything Constantly Available
Logical Connection
A
B
Everything Conditionally Available
B
Busy
B
Not Available
Queue
A
A
B
B
Not Available
Not Available
A
Not Available
A
Message/Queuing Environment

What is a message?

What's a Message?

Message = Header + User Data



A Series of Message Attributes Understood and augmented by the Queue Manager
- Unique Message Id
- Correlation Id
- Routing information
- Reply routing information
- Message priority
- Message codepage/encoding
- Message format
- ....etc.

Any sequence of bytes
- Private to the sending and receiving programs
- Not meaningful to the Queue Manager

Message Types

- Persistent ... recoverable
- Non Persistent

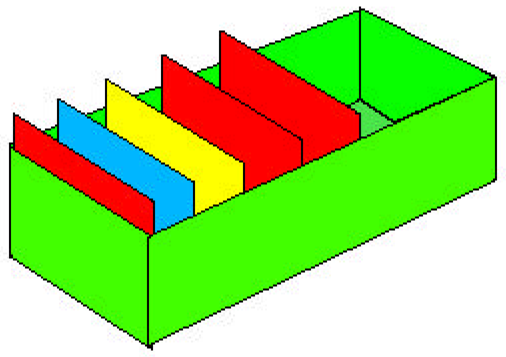
Up to 100MB message length

What's a Queue?

Place to hold messages

Queue creation

- Pre-defined
- Dynamic definition



Message Access

- FIFO
- Priority
- Direct
- Destructive & non-destructive access

Parallel access by applications

- Managed by the queue manager

FEDERAL STUDENT AID
We Help Put America Through School

A Queue manager is made of 3 components

A queue manager may - generally - be thought of as 3 components:

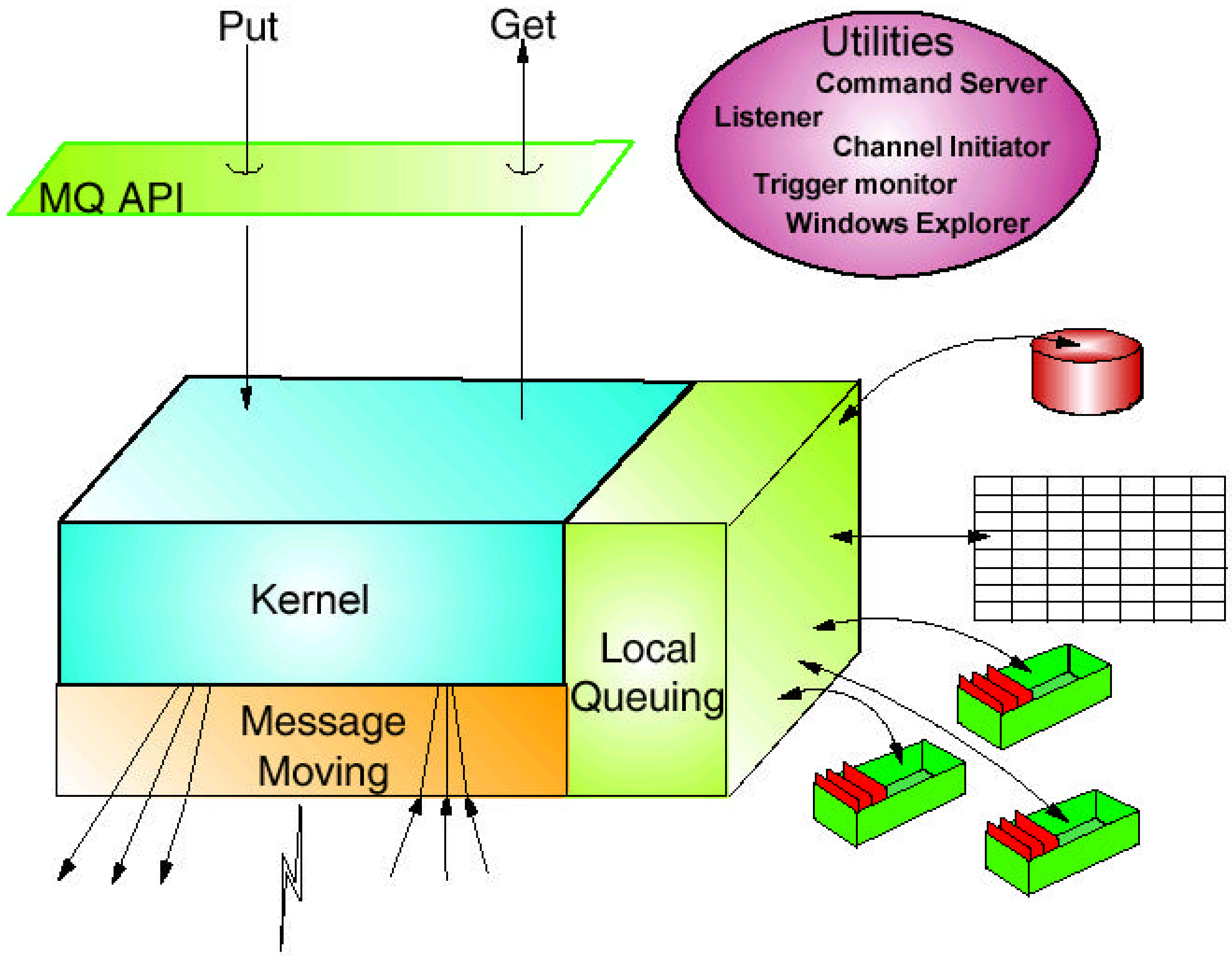
- The Kernel is the part of the queue manager that understands how to implement the MQSeries APIs. Given that the APIs are common across the queue manager family, it stands to reason that the Kernel is mostly common code across the set of queue managers. (The primary exception to this is the OS/390 queue manager where the same functions are implemented differently to support the same APIs).

- The Local queuing component is the part of the queue manager responsible for interacting with the local operating system. It manages memory, the file system and any operating system primitives such as timers, signals, etc. This component insulates the Kernel from any considerations of how the underlying operating system provides services and so enables the Kernel to be operating system independent.

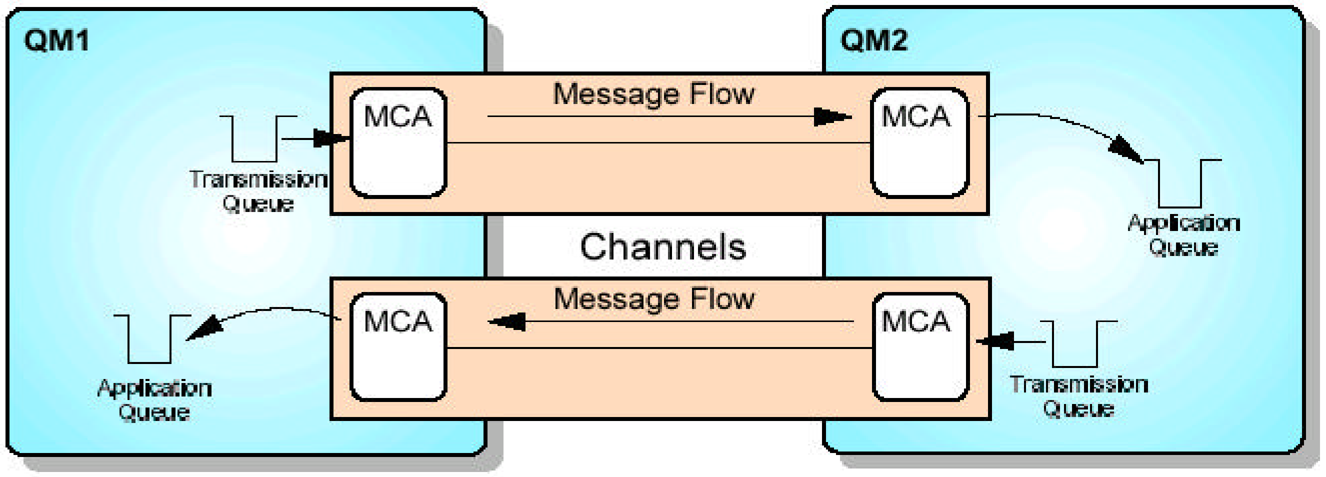
- The Message Moving component is responsible for interacting with other queue managers and with MQI clients. For environments where all of the message queuing activity is local to a system then this component is unused - though this is a very rare case.
  The message moving functions are provided by specialised MQSeries applications, called **M**essage **C**hannel **A**gents.

FEDERAL STUDENT AID
We Help Put America Through School

What is a Queue Manager?



Channel Concepts



Channels are uni-directional
Channels provide for (application) session concentration
Two-way communication requires two channels

Channel Concepts Notes

Channels are used by MQSeries Queue Managers in order to exchange messages between Queue Manager implementations. This chart illustrates the various components that are required for an assured message delivery mechanism.

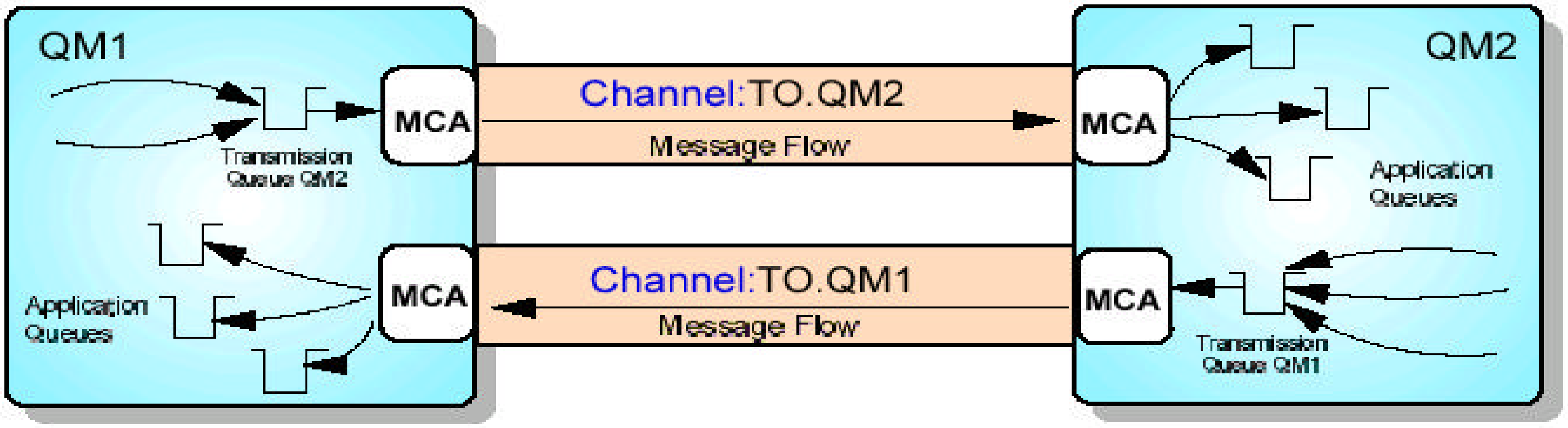
- When an MQ application opens a queue, it is necessary to have a queue name resolution process which will enable the appropriate destination to be targeted. In this case, the target destination is a remote queue - which resolves to a local transmission queue
- There must be a local mechanism for safe storage of messages until they can be passed to the target Queue Manager and queue. This is the transmission queue. This queue is the same as any local queue - with the exception that the usage of the queue is designated as 'xmitq'. Note that transmission queue is often abbreviated to xmitq. Note that ALL messages destined for a remote Queue Manager must pass through a Transmission Queue.
- There must be a process that is responsible for taking messages from the transmission queue and passing them to the remote Queue Manager using some underlying (provided) transport system. This process is called the sending Message Channel Agent (MCA).
- There must be a process that is responsible for receiving messages and placing them onto target queues. This process is known as the receiving MCA.

A sending and receiving MCA pair - and the underlying transport are known as an MQSeries channel. There is a transport independent protocol defined for MCAs to facilitate the once-only, assured delivery of messages.

FEDERAL STUDENT AID
We Help Put America Through School

Channel Definitions

QM1
DEF CHL(TO.QM1) CHLTYPE(RCVR)
DEF QL(QM2) USAGE(XMITQ)
DEF CHL(TO.QM2) CHLYPE(SDR) TRPTYPE(TCP) CONNAME(QM2.HURSLEY) XMITQ(QM2)



QM2
DEF CHL(TO.QM2) CHLTYPE(RCVR)
DEF QL(QM1) USAGE(XMITQ)
DEF CHL(TO.QM1) CHLYPE(SDR) TRPTYPE(TCP) CONNAME(QM1.HURSLEY) XMITQ(QM1)

FEDERAL STUDENT AID
We Help Put America Through School

Channel Format and Protocol

Once/once-only message delivery

Resynchronisation process for failed channel

A channel is a one-way conduit between 2 Queue Managers. It provides a single pipe (session) through which all messages bound for a particular partner Queue Manager may be sent. This is in contrast to some mechanisms which require a pipe (session) per application. Because the channel is a one-way mechanism, MQSeries messages may flow in one direction only. If two directional flow is required between Queue Managers then two channels are required.

Further, it is important to understand that although message flow for a channel is uni-directional, there are control packets flowing in both directions. Messages are not required to be passed directly to their target queue. It is quite acceptable for a message to be passed through one (or many) intermediate Queue Managers before reaching the final destination. This is simply achieved by defining appropriate transmission queues on the intermediate Queue managers.

The receiving message channel agent will place messages destined for remote Queue Managers onto these transmission queues. The method by which this is accomplished is explained later. This method of passing messages is usually known as multi-hop.

It is therefore possible to construct any topology of interconnected Queue Managers. Perhaps the most popular forms being :-

'any-to-any'	where every Queue Manager is directly connected to every other Queue Manager
'hub'	where a central Queue Managers routes messages to other Queue Manager.

Three-tier networks where MQI applications connect in to one of the leaf node Queue Managers via the client facility are also very popular.

FEDERAL STUDENT AID
We Help Put America Through School

MQSeries Messaging Platforms

Servers:

- OS/390
- AIX
- Windows NT, 2000
- Solaris: Intel & SPARC
- HP-UX
- OS/400
- OS/2
- Compaq OpenVMS
- Compaq NSK
- Compaq Tru64 UNIX
- VSE/ESA
- Windows: 3.1,95,98
- Digital UNIX
- SunOS

✚ SCO: Openserver, UnixWare
✚ IRIX
- Dynix/ptx
- NCR
- TPF

✚ DC/OSx
- Sinix
- Linux
- PalmOS (MQe)
- EPOC (MQe)
- Java (MQe)

★ Unisys 2200
★ Hitachi

Clients only:

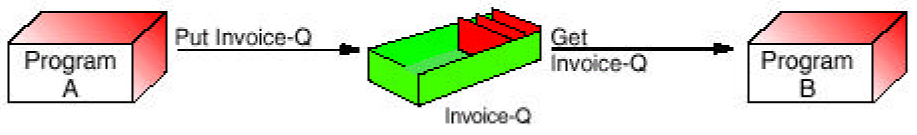
✚ DG/UX
✚ HP3000 MPE/iX
- Java
- Windows: 3.1,95,98
- DOS
- VM

✚ Apple MacOS
✚ Stratus VOS
✚ 4690
✚ Unisys A

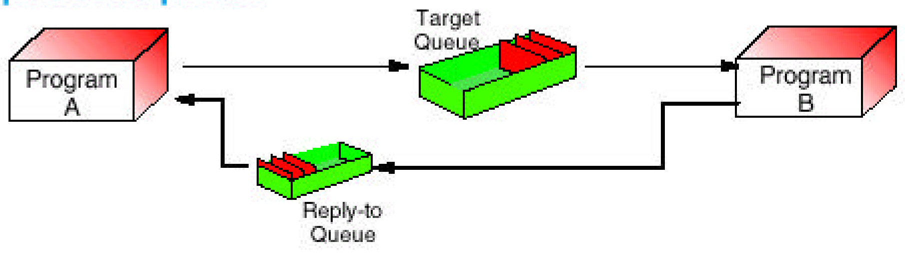
39 Platforms

Examples

'Fire and Forget'

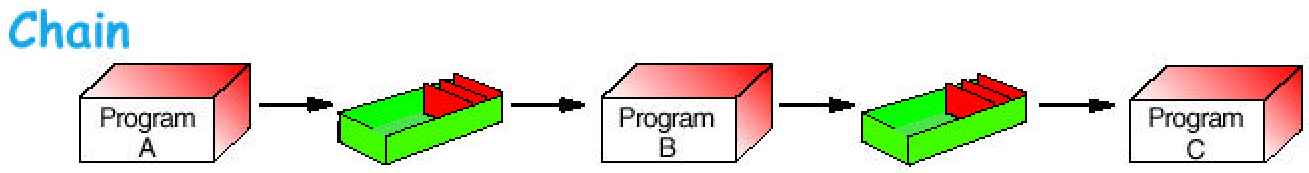


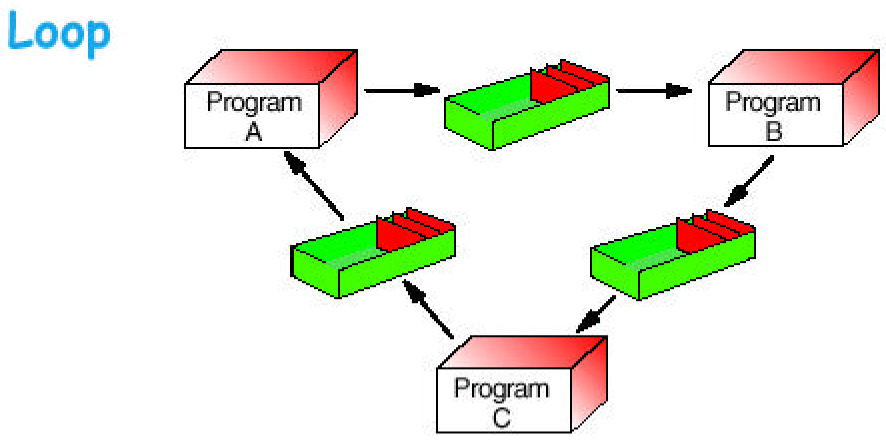
Request/Response



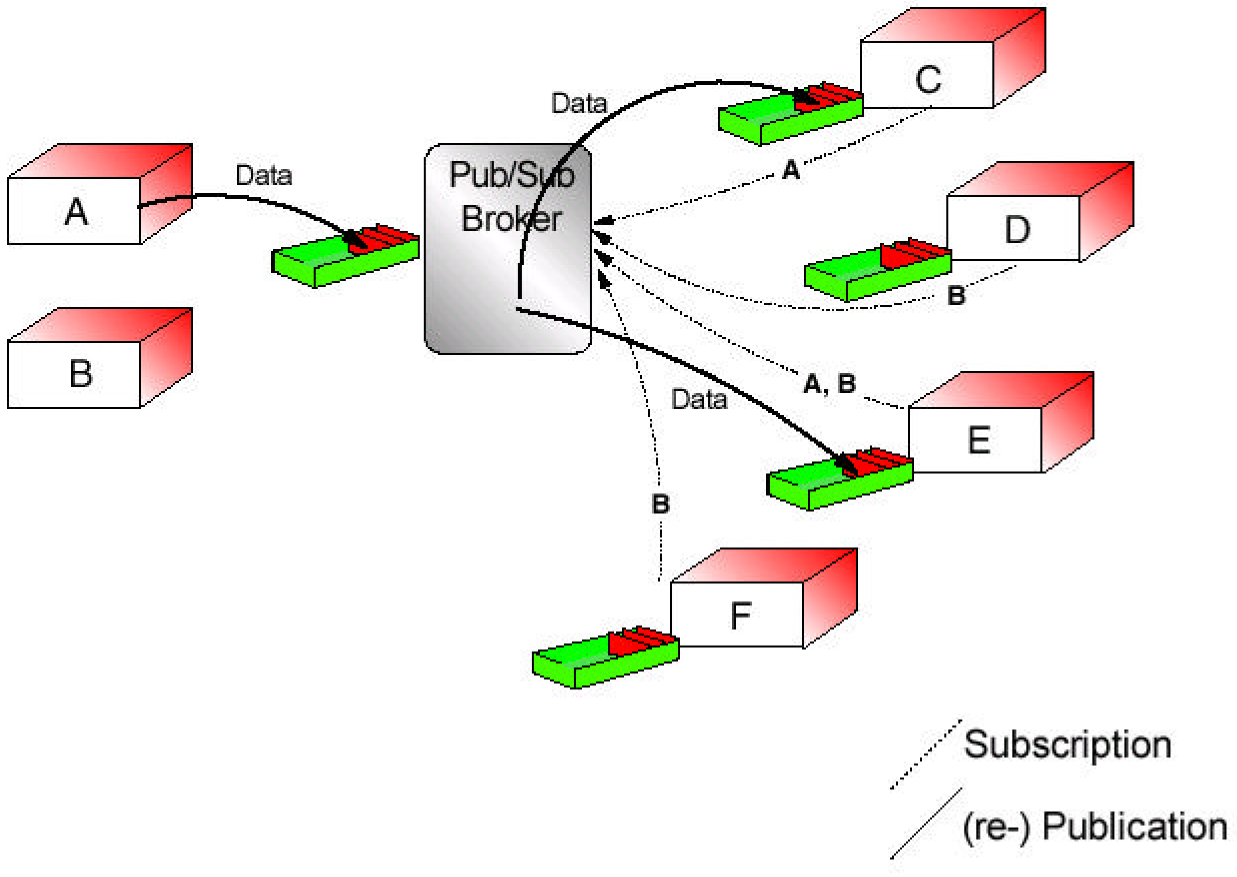
FEDERAL STUDENT AID
We Help Put America Through School

Examples



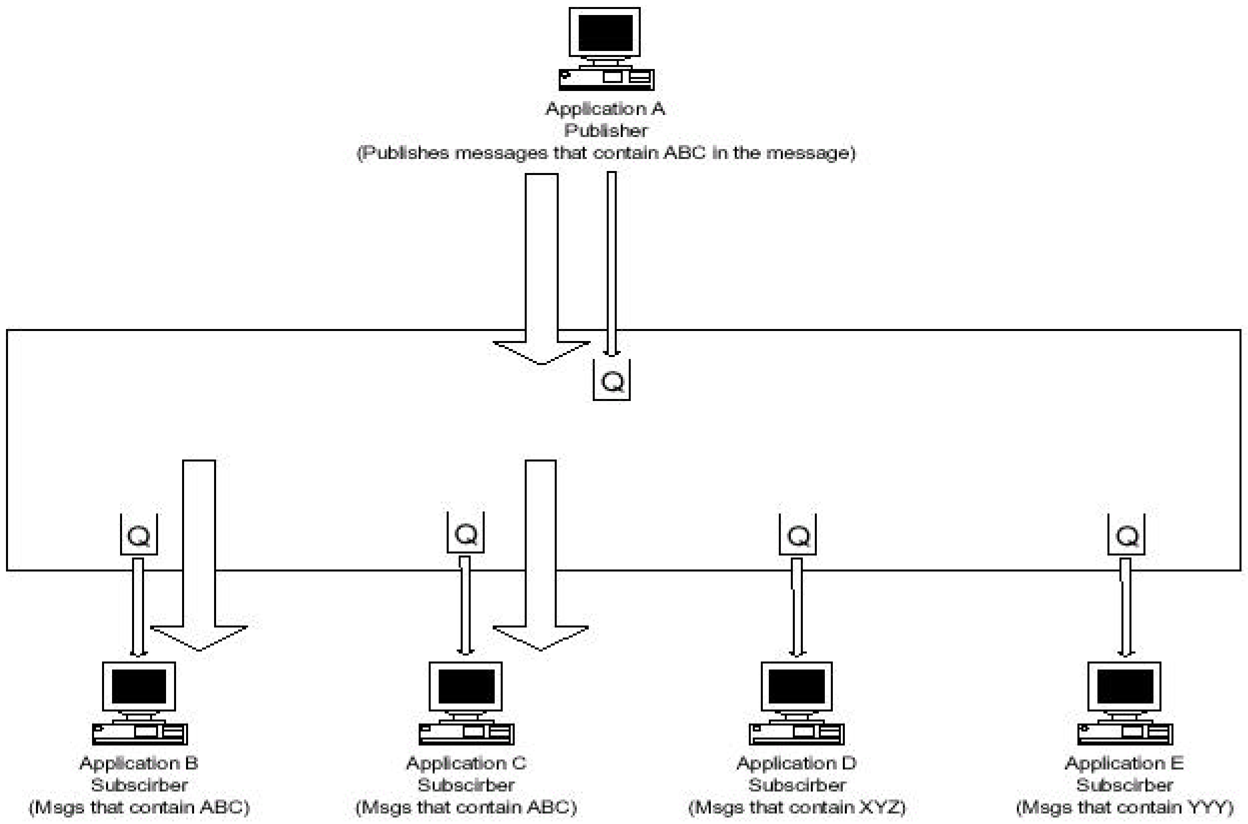


MQSeries Publish / Subscribe



FEDERAL STUDENT AID
We Help Put America Through School

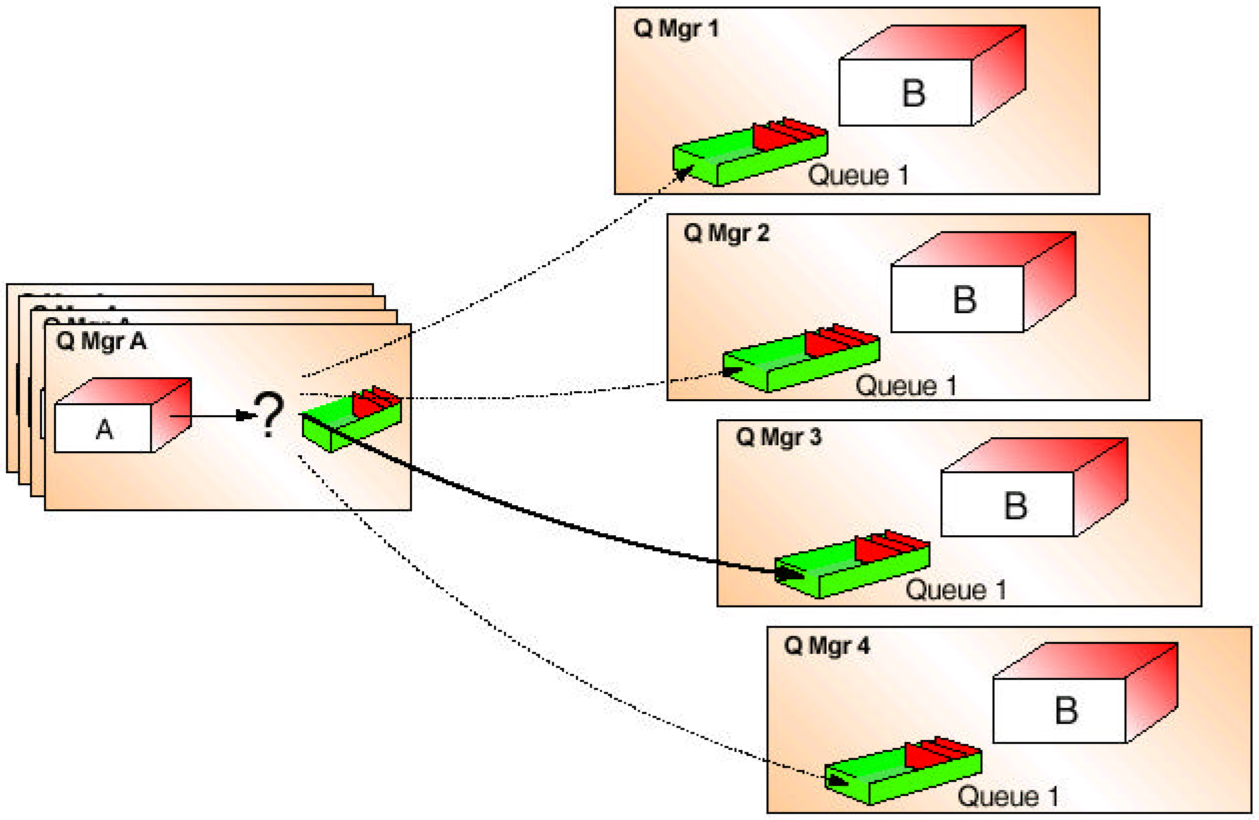
Bus – MQSeries Integrator Pub/Sub



- Key Points
  - Based on MQSeries base as well as the MQSeries Integrator products thus deriving the MQSeries base and MQSI products' strengths
  - Takes administration of application to application communication out of the applications realm and into the administrative domain
  - Centrally administered

FEDERAL STUDENT AID
We Help Put America Through School

Queue Manager Clusters



FEDERAL STUDENT AID
We Help Put America Through School

Pub/Sub & Clusters Notes

The final example given here (though not the last possibilitiy by any means) is **MQSeries and publish/subscribe**. In this environment, the receiving applications notify an intermediate broker of their interest in particular sets of information...in this context a receiving (or subscribing) application provides a subject and a queue where messages matching this subject may be delivered. When sending (publishing) applications generate information they also provide an associated subject and the broker provides a matching service enabling only the appropriate subscribing applications to receive the information. Note that the publish/subscribe model provides for the situation where a message may be published by an application on a subject which has no subcribers. In this instance the message data is discarded.

There are many publish/subscribe products available in the marketplace today. MQSeries publish/subscribe differentiates itself by providing support for the publish/subscribe model and combining it with the exactly once delivery model of MQSeries message/queuing.

In order to enable highly scalable applications, MQSeries queue managers provide support for **MQSeries Clusters**. In this environment, there are several copies (or clones) of a particular target queue and each message is sent to *exactly one* of the possible choices. MQSeries Cluster support also defines and manages all MQSeries resources automatically and provides automatic notification of failed or new queue managers in the environment.

FEDERAL STUDENT AID
We Help Put America Through School

Examples: Notes

These examples illustrate some of the ways in which MQSeries queues can be used and, thereby, illustrates some of the styles of applications that may benefit from the use of a message/queuing model.

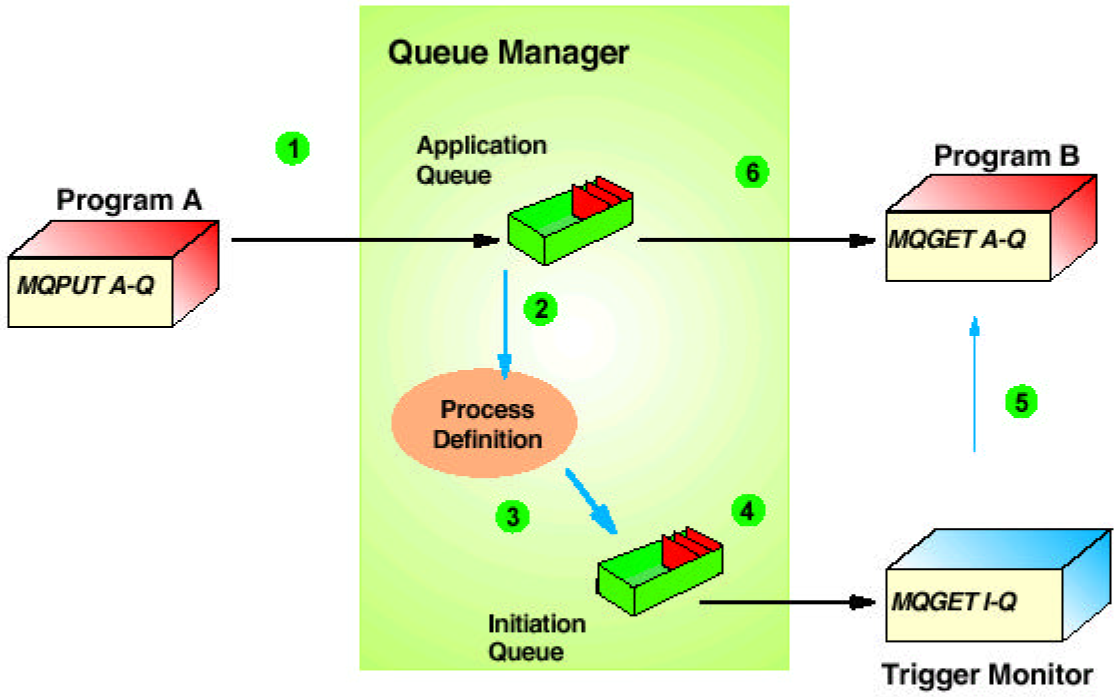
- The '**fire and forget**' style of operation is one where there is no (direct) response required to a message. The message/queuing layer will guarantee the arrival of the data withouth the application having to solicit a response from the receiver.
- The **request/response** style is typical of many existing synchronous applications where some response is required to the data sent. This style of operation works just as well in an asynchronous environment as in a synchronous one. One difference is that - in this case - the sender does not have to wait for a response immeditely. It could pick up the response at some later time in it's processing. Although this is also possible with the synchronous style, it is less common.
- Data does not have to be returned to the originating application. It may be appropriate to pass a response to some other application for processing, as illustrated in a **chain** of applications.
- There may be multiple applications involved in the processing before a response comes back to the originating application, giving a **loop** of applications.

These various modes of interaction may be arbitarily combined to provide as complex/sophisticated a topology as is necessary to support a particular application. The loosley coupled nature of the message queuing model makes it ideal for this style of interaction. Further, it makes it straightforward to develop applications in an iterative style.

In order to enable highly scalable applications, MQSeries queue managers provide support for **MQSeries Clusters**. In this environment, there are several copies (or clones) of a particular target queue and each message is sent to *exactly one* of the possible choices. MQSeries Cluster support also defines and manages all MQSeries resources automatically and provides automatic notification of failed or new queue managers in the environment.

FEDERAL STUDENT AID
We Help Put America Through School

Triggering



Triggering allows

- Instantiation as required
- Conservation of system resources
- Automation of flow

Triggering Nodes

The time-independent nature of message queuing means that applications may be idle during periods when there are no messages to process. To avoid having processes consuming system resources while there is no work to do, MQSeries provides a mechanism to 'trigger' applications to start - when certain conditions are met.

Triggering works by defining a condition for an application queue which, when met, causes the queue manager to send a trigger message to an initiation queue. The trigger message is a fixed-format message which contains information about the application queue that caused the trigger message to be generated and the application which should be started to process messages. Trigger messages are processed by a Trigger Monitor which (usually) runs continuously. The Trigger Monitor starts the appropriate application to process the message(s) on the application queue.

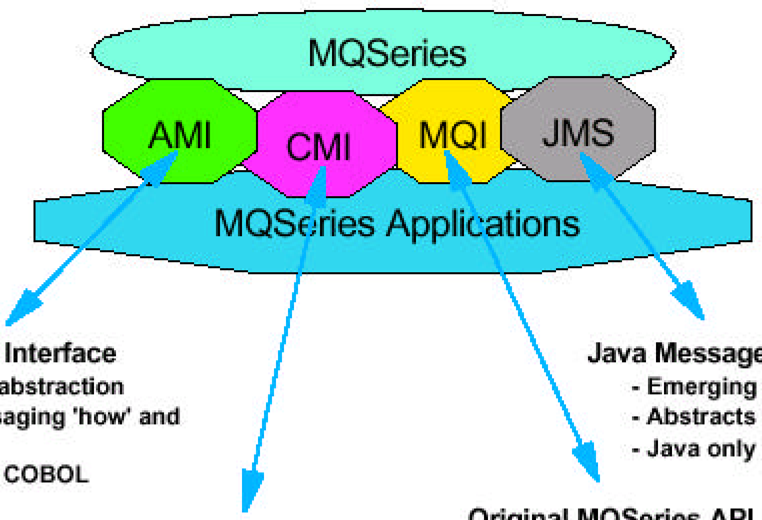
Trigger conditions can be based on queue depth, for all messages or for messages above a certain priority. Or a trigger message could be generated for every message or just for the first message on a queue. Further, a minimum time interval may be specified between trigger events which avoids any issues with duplicate triggering.

By using a Trigger Monitor it is possible to have a single process which initiates many application processes to handle messages arriving on one or many queues as required.

MQSeries provides basic Trigger Monitors for many environments. However, because of the diversity of environments, it may be necessary to implement a *User-written* Trigger Monitor. The triggering process is entirely exposed and documented, in order to make this possible.

Triggering is not the only mechanism available to start applications - and it is not always the most appropriate solution. But, is is one option to be considered.

MQSeries Interfaces



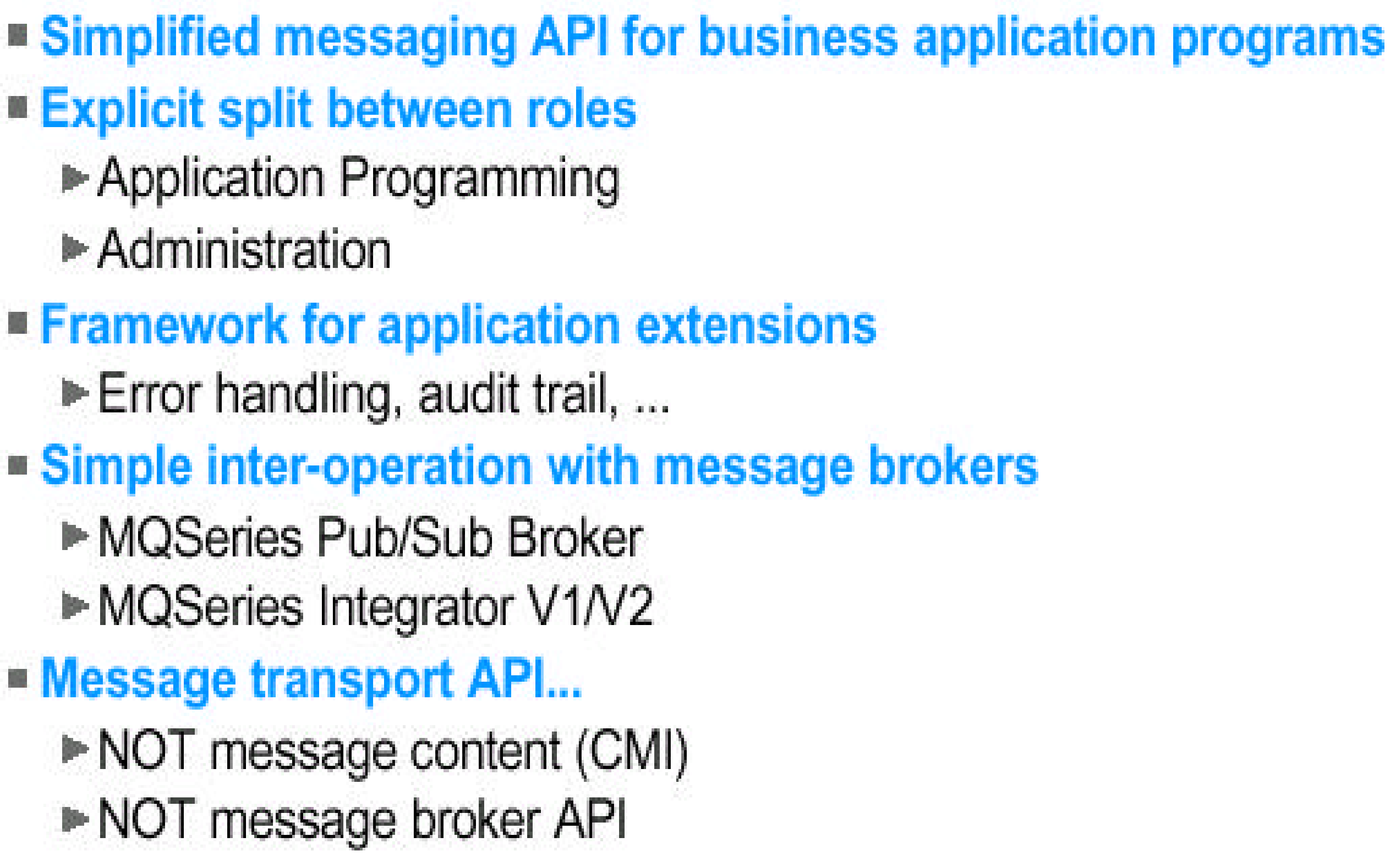
\pp" Messaging Interface
- High level of abstraction
- Defines messaging 'how' and 'where'
- C, C++, Java, COBOL

Java Message Service
- Emerging Java standard
- Abstracts MQSeries specific
- Java only

Common Message Interface
- Construction and analysis of messages using logical view
- Supports XML, COBOL copybook, C header
- Usable with AMI, JMS, MQI
- Not yet available

Original MQSeries API
- Continued support
- 390 Assembler, C, C++, COBOL, COM, LotusScript, Java, PL/1, VisualBasic

AMI Objectives



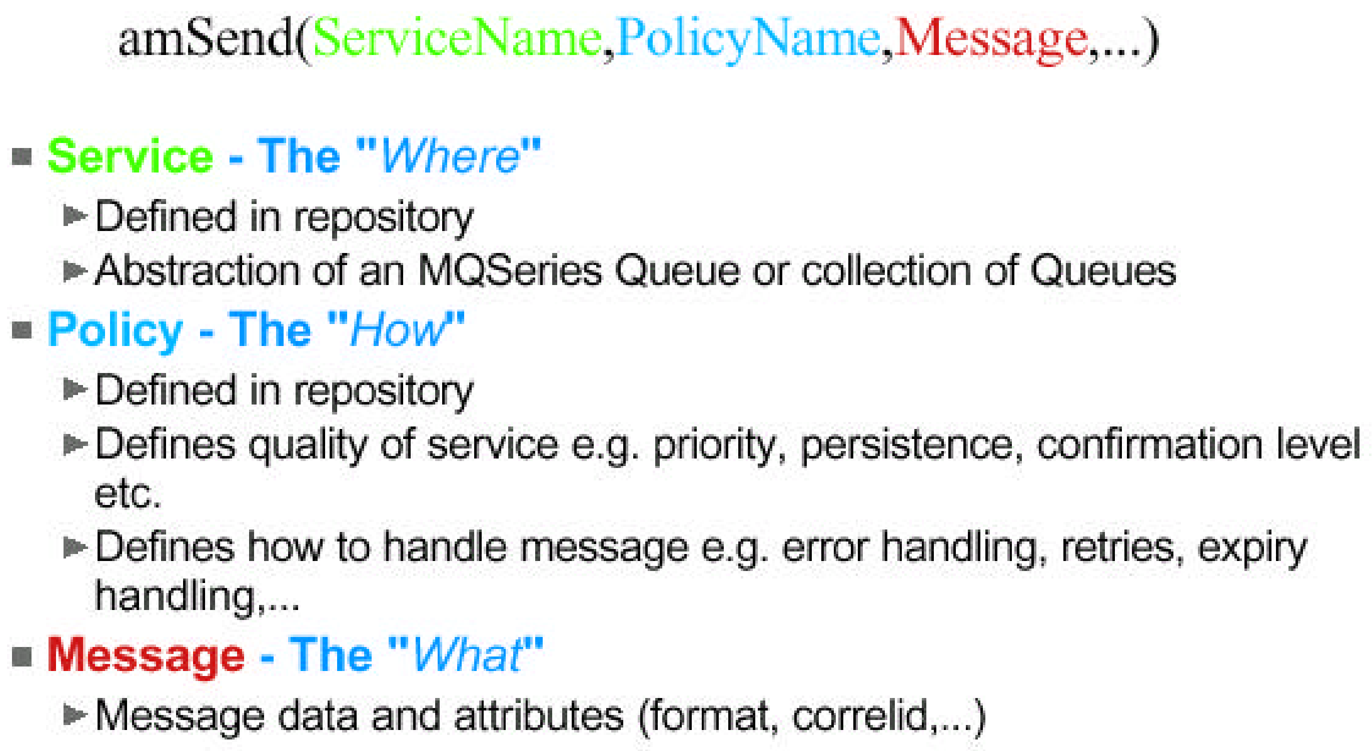
- **Simplified messaging API for business application programs**
- **Explicit split between roles**
  - Application Programming
  - Administration
- **Framework for application extensions**
  - Error handling, audit trail, ...
- **Simple inter-operation with message brokers**
  - MQSeries Pub/Sub Broker
  - MQSeries Integrator V1/V2
- **Message transport API...**
  - NOT message content (CMI)
  - NOT message broker API

Philosophy

- Simplified interface
- Fewer structures, more verbs with single function
- Message handling/transport behaviour options moved from program to administration domain (policies)
- Separate calls for different application styles
  - Send and Forget
  - Request/Response
  - Request/Reply
  - Publish and Subscribe
  - File Transfer
- Natural style for each language, C, C++, Java

FEDERAL STUDENT AID
We Help Put America Through School

AMI Concepts



amSend(ServiceName,PolicyName,Message,...)

- **Service - The "*Where*"**
  - Defined in repository
  - Abstraction of an MQSeries Queue or collection of Queues
- **Policy - The "*How*"**
  - Defined in repository
  - Defines quality of service e.g. priority, persistence, confirmation level etc.
  - Defines how to handle message e.g. error handling, retries, expiry handling,...
- **Message - The "*What*"**
  - Message data and attributes (format, correlid,...)

AMI Procedure Style Calls

- **Setup**
  - amInitialize()
  - amTerminate()
- **Datagram**
  - amSendMsg()
  - amReceiveMsg()
- **Request/Reply**
  - amSendRequest()
  - amReceiveMsg()
- **Request/Response**
  - amReceiveRequest()
  - amSendResponse()
- **Publish/Subscribe**
  - amPublish()
  - amReceivePublication()
  - amSubscribe()
  - amUnsubscribe()
- **File Transfer**
  - amSendFile()
  - amReceiveFile()

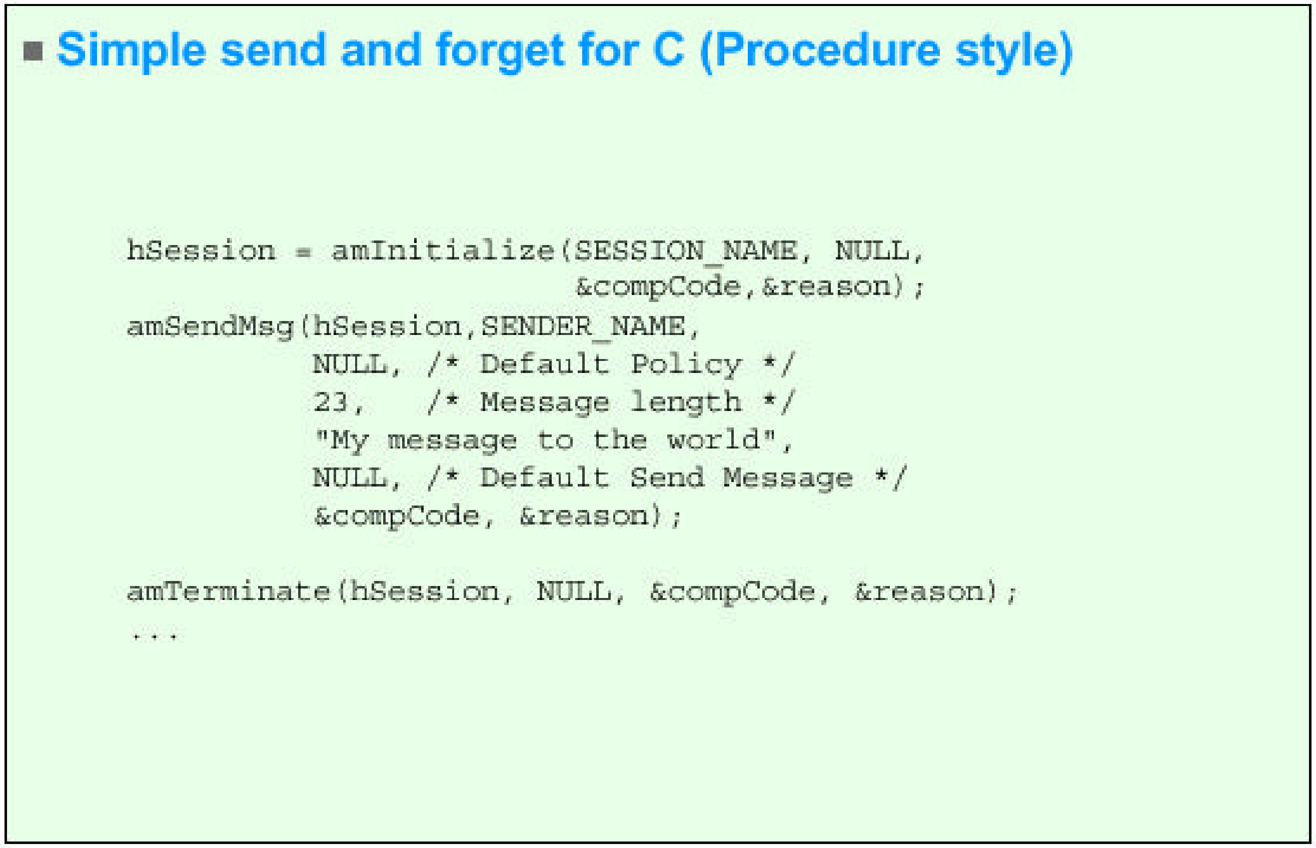
FEDERAL STUDENT AID
We Help Put America Through School

AMI Object Style Calls

- **Grouped by "*Object Type*"**

Session:	amSesXXX()	
Policy:	amPolXXX()	
Message:	amMsgXXX()	
Sender:	amSndXXX()	[Service]
Receiver:	amRcvXXX()	[Service]
DistributionList:	amDstXXX()	[Service]
Publisher:	amPubXXX()	[Service]
Subscriber:	amSubXXX()	[Service]

AMI API Examples



- **Simple send and forget for C (Procedure style)**

```
hSession = amInitialize(SESSION_NAME, NULL,
                        &compCode,&reason);
amSendMsg(hSession,SENDER_NAME,
          NULL, /* Default Policy */
          23,   /* Message length */
          "My message to the world",
          NULL, /* Default Send Message */
          &compCode, &reason);

amTerminate(hSession, NULL, &compCode, &reason);
...
```

AMI API Examples

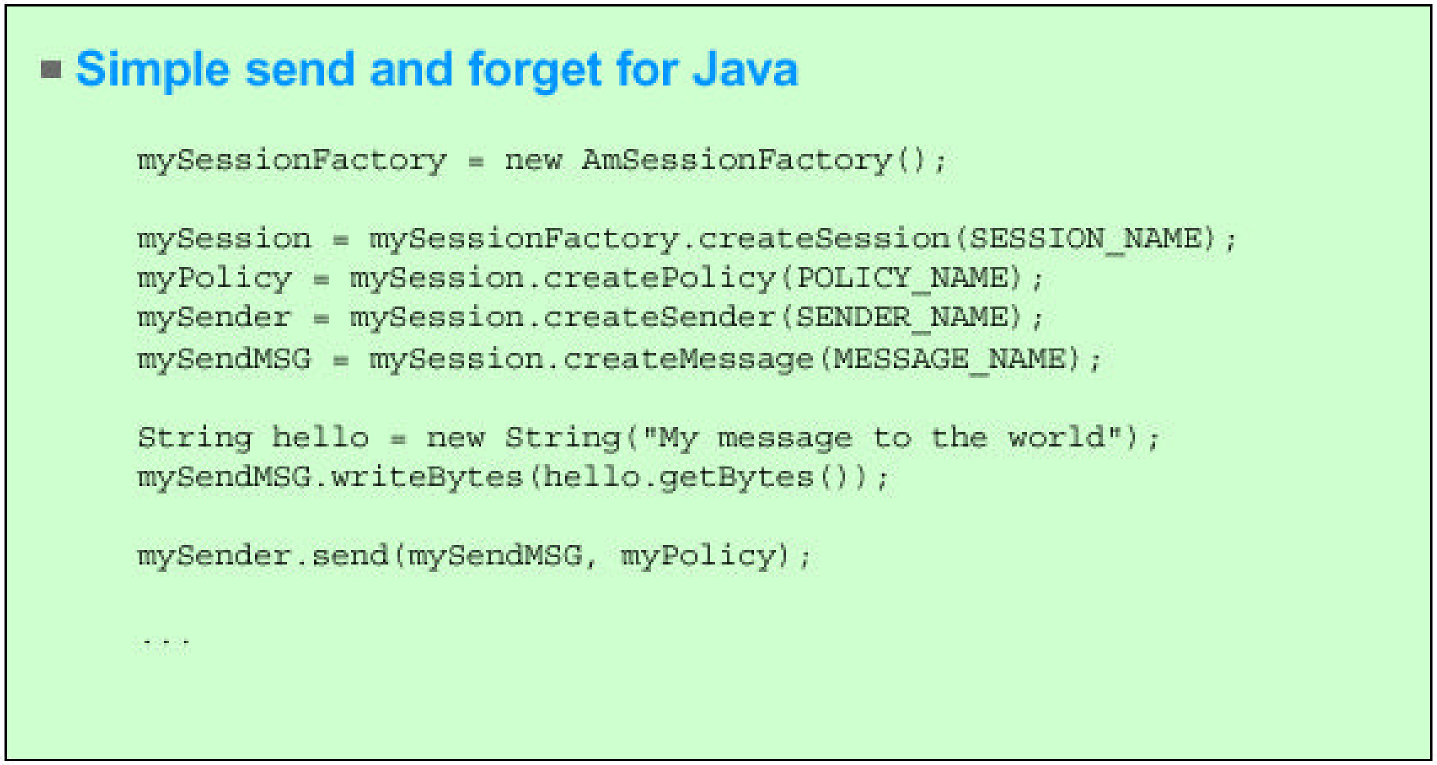
- Simple send and forget for C (Object style)

```
hSession = amSesCreate(SESSION_NAME,&compCode,&reason);
hSender = amSesCreateSender(hSession,SENDER_NAME,
                            &compCode, &reason);
hMsg = amSesCreateMessage(hSession,SENDER_NAME,
                            &compCode, &reason);

amSesOpen(hSession,AMH_NULL_HANDLE,&compCode,&reason);
amSndOpen(hSender,AMH_NULL_HANDLE,&compCode,&reason);
amSndSend(hSender,AMH_NULL_HANDLE,AMH_NULL_HANDLE,
                  AMH_NULL_HANDLE, 23,
                  "My Message to the world",
                  hMsg, &compCode, &reason);
...
```

FEDERAL STUDENT AID
We Help Put America Through School

AMI API Examples



- Simple send and forget for Java

```
mySessionFactory = new AmSessionFactory();

mySession = mySessionFactory.createSession(SESSION_NAME);
myPolicy = mySession.createPolicy(POLICY_NAME);
mySender = mySession.createSender(SENDER_NAME);
mySendMSG = mySession.createMessage(MESSAGE_NAME);

String hello = new String("My message to the world");
mySendMSG.writeBytes(hello.getBytes());

mySender.send(mySendMSG, myPolicy);

...
```

AMI Objects

- Session
  - Creates and manages all other objects
  - Owns the connection object
  - Provides the scope for a unit-of-work
- Connection
  - Establishes message transport (MQSeries) connection
  - Defined in AMI tool
  - Not directly visible to an application

AMI Objects

- **Message**
  - Encapsulates the message attributes (priority etc.)
  - Optionally stores the message data (if not passed in an application buffer)
- **Policy**
  - Encapsulates options for open(), close(), send(), receive(), publish(), subscribe() etc.
  - Can be defined in repository, built-in or default definition
- **Service**
  - Encapsulates one (or in the case of a distribution list more than one) MQOD structure that references a local or non-local MQSeries Queue

FEDERAL STUDENT AID
We Help Put America Through School

AMI Repository

- Optional (AMI can work with or without repository)
- Can define:
  - Policy
    - Connection: name, type: real/logical, mode: auto/client/server
  - Service Points
  - Services
    - Distribution List*
    - Publisher
    - Subscriber

* a Distribution List requires a repository definition (defaults exists for other types)

AMI Built-In Policy Behavior

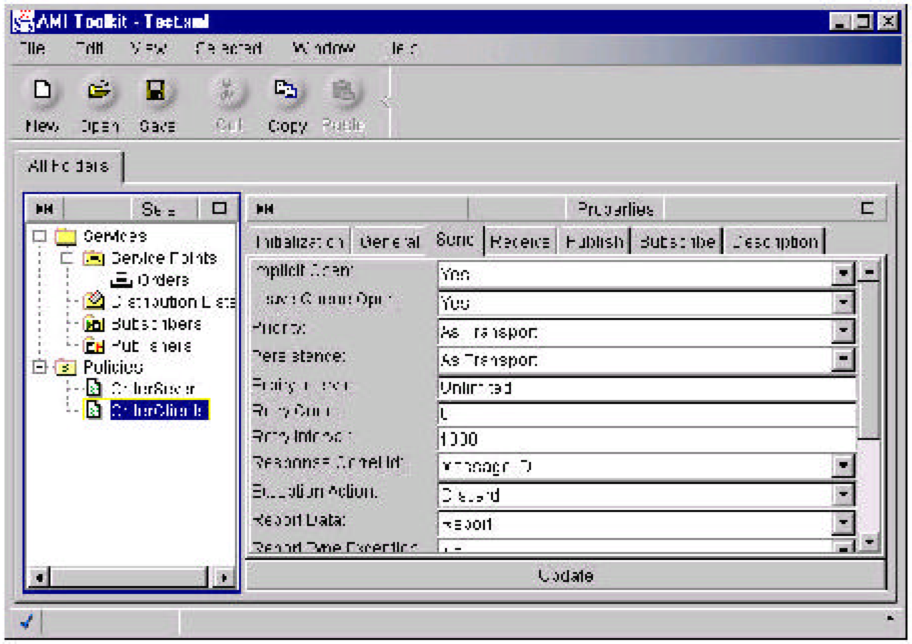
- Pre-defined behaviour by using a particular policy
- Cuts down on application code
- For example, poison message handling
  - When backout-limit is exceeded, poison message is requeued to backout-requeue Queue and MQRC_BACKOUT_LIMIT_ERR returned
  - If requeue fails, message is returned to application with MQRC_BACKOUT_REQUEUE_ERR
  - If message is part of group, no attempt is made to requeue and MQRC_GROUP_BACKOUT_LIMIT_ERR is returned

FEDERAL STUDENT AID
We Help Put America Through School

AMI Repository

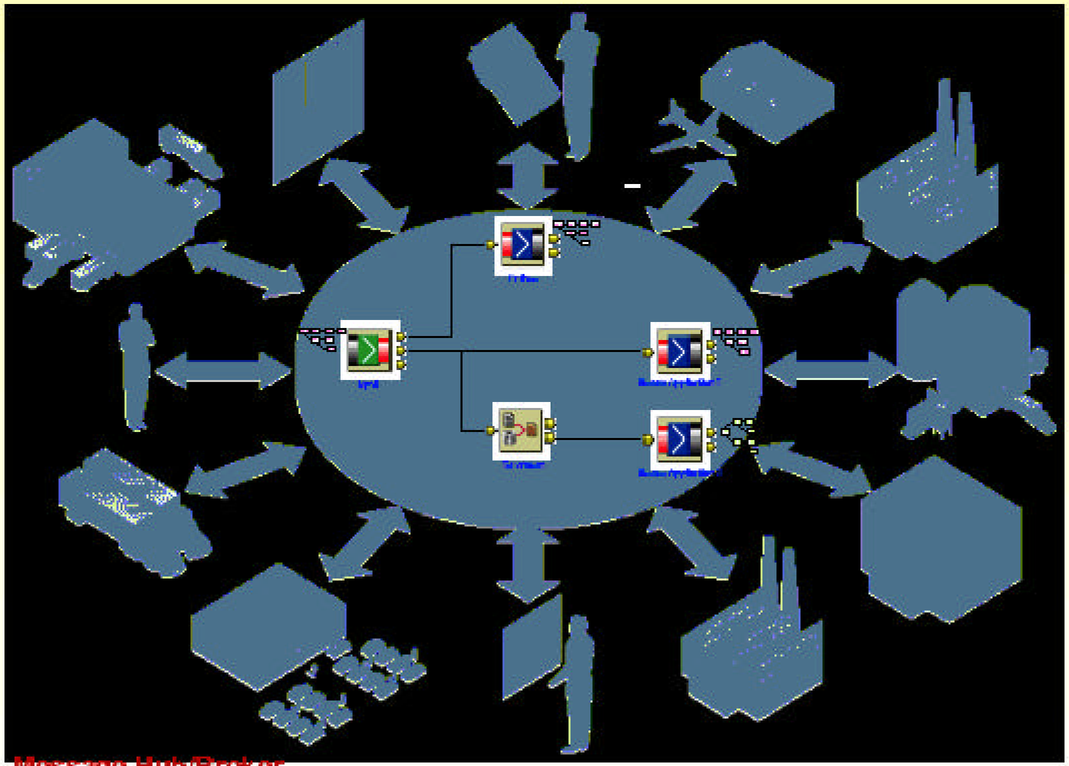
- **Build Time**
  - Java based GUI tool
  - NT Only
  - Produces XML repository file
- **Run Time**
  - AMI reads XML repository file
  - Each application can have a different XML file
    - Use of environment variable `AMT_REPOSITORY` for C
    - Use of `setRepository()` method in C++/Java
  - XML files can be shared using standard file sharing

AMI Administration Tool



Implicit Open:	Yes
[illegible] Opt:	Yes
Priority:	As Transport
Persistence:	As Transport
Expiry Interval:	Unlimited
Retry Count:	0
Retry Interval:	1000
Response Correl Id:	Message ID
Exception Action:	Discard
Report Data:	Report
Report Type Exception:	[illegible]

Creating an Application Integrator



Message Hub/Broker

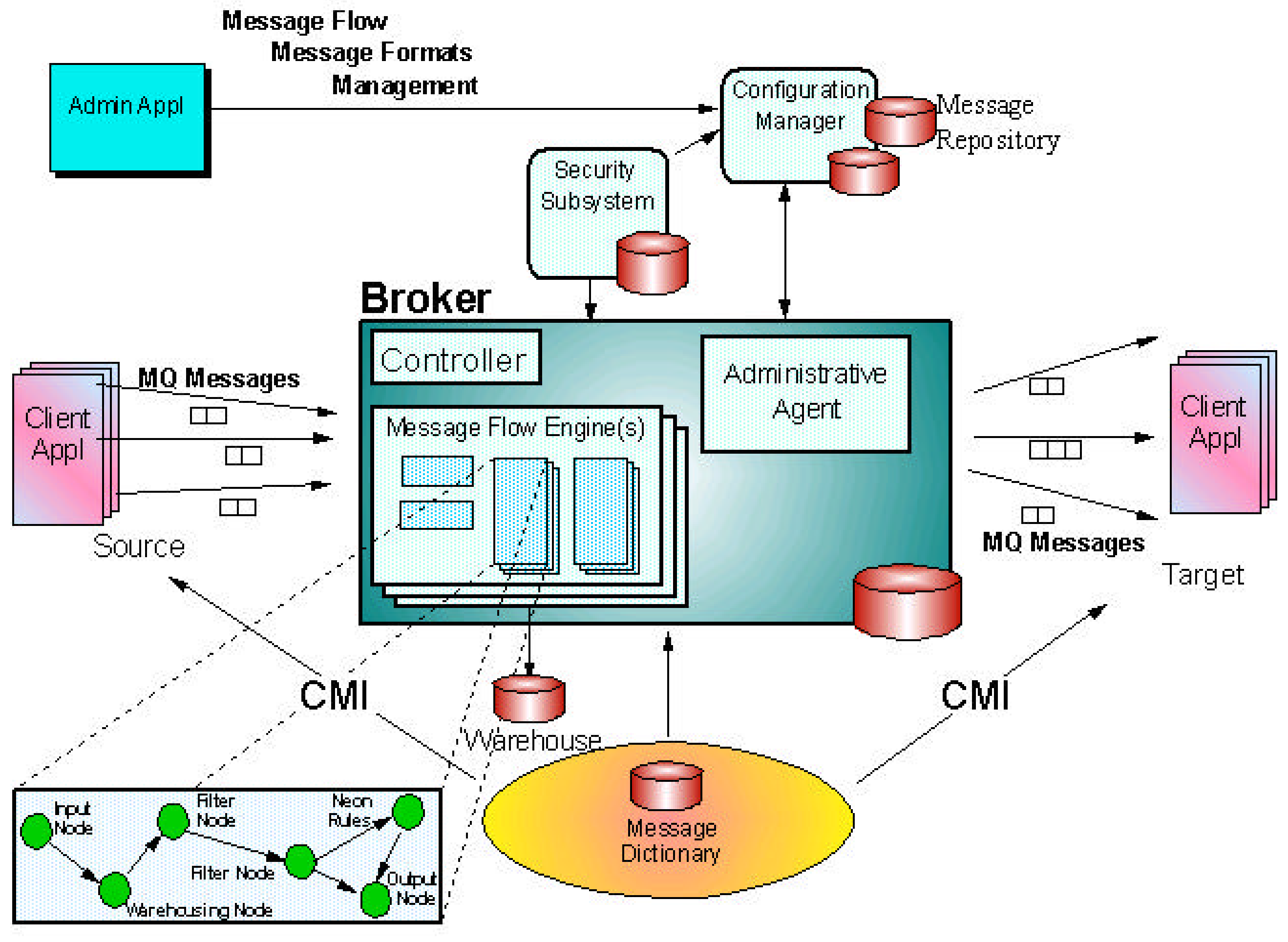
- Transformation
- Business Rules
- Intelligent Routing

- ✓ Join Applications & Information sources
- ✓ Heterogeneous & decoupled
- ✓ Data transform
- ✓ Data routing
- ✓ DBMS Integration
- ✓ Transactional
- ✓ Tooling

- ✓ Simple
- ✓ Extensible
- ✓ Standards based

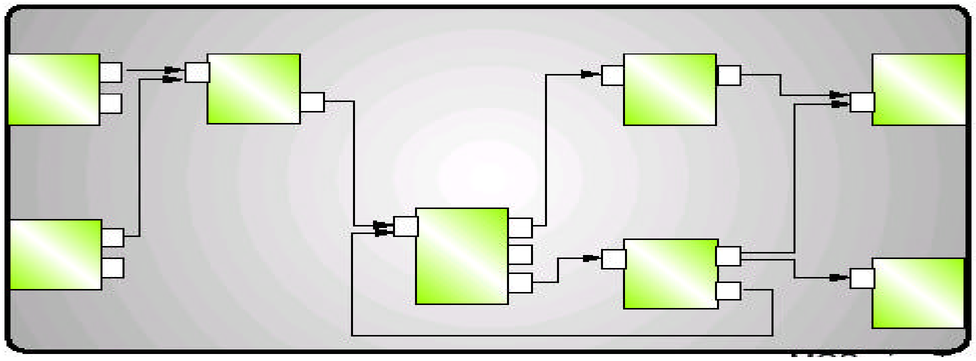
FEDERAL STUDENT AID
We Help Put America Through School

MQSI Architecture



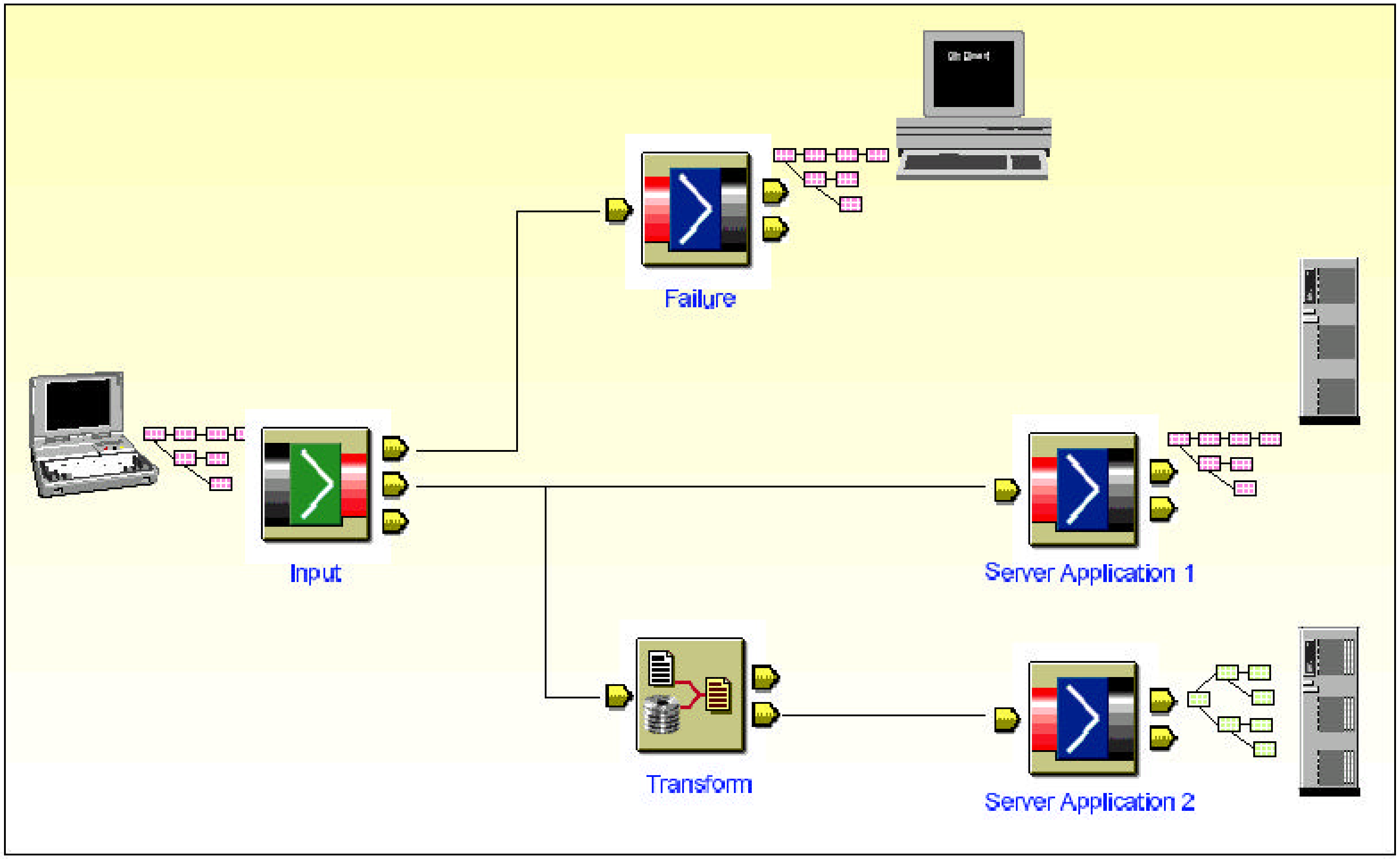
Message Flow Components

- **A message Flow is**
  - A sequence of operations on a message
  - Dependent on message content and current message flow actions
  - Stateless across successive inputs
  - Constructed using visual "wiring" tool from Process Nodes
- **A message flow is composed of**
  - Built-in IBM supplied processing nodes
  - 3rd party "plug-in" processing nodes
    - Defined interface for processing nodes
  - Process nodes which are previously defined message flows
    - Allows building compound message flows
- **Message Flows can be thought of as "Business Services"**



FEDERAL STUDENT AID
We Help Put America Through School

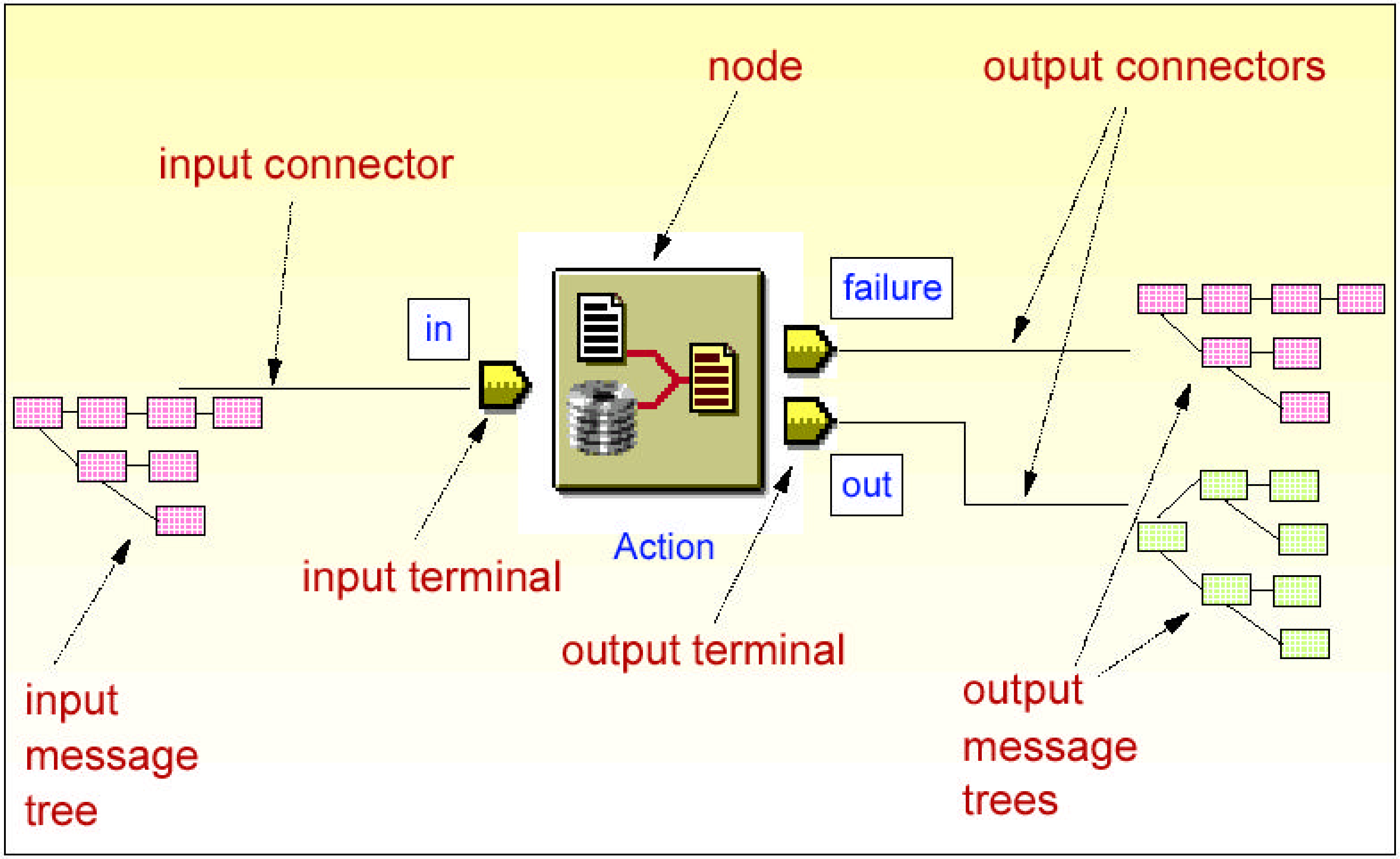
MQSI: Message Flows



FEDERAL STUDENT AID
We Help Put America Through School

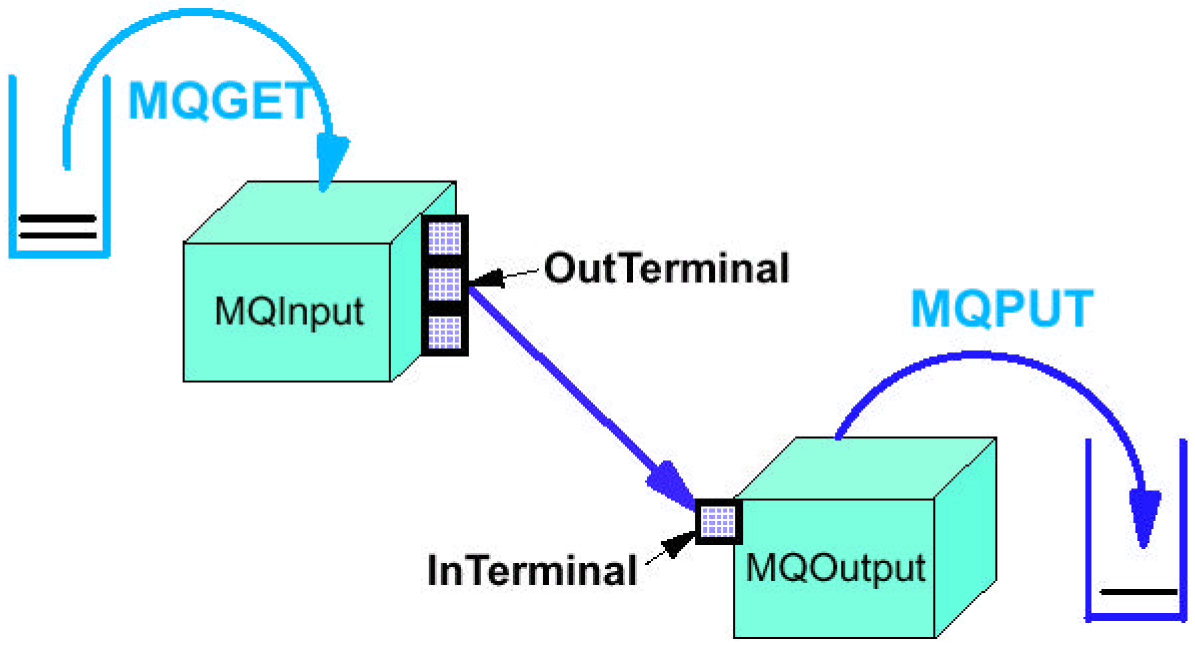
MQSI: Nodes

Message Processing Nodes



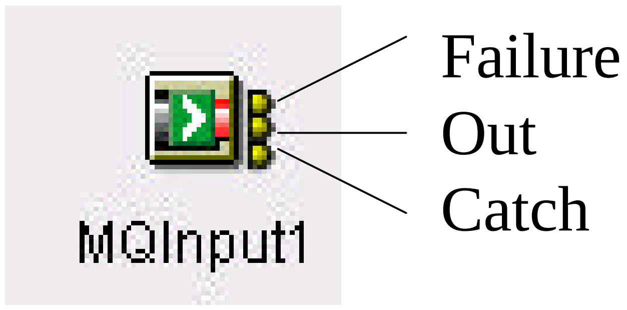
MQSI: Generic Flow

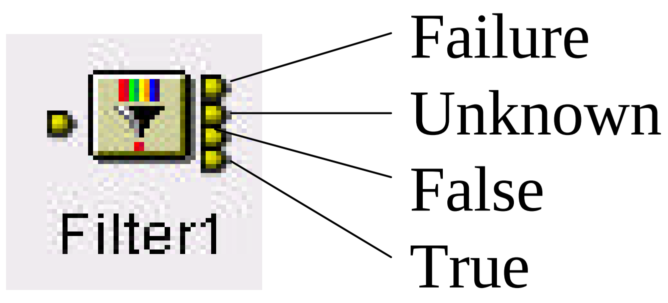
A Simple Message Flow



MQSI: Terminals

Each node has one input terminal and one or more output terminals





Message Flow Nodes...

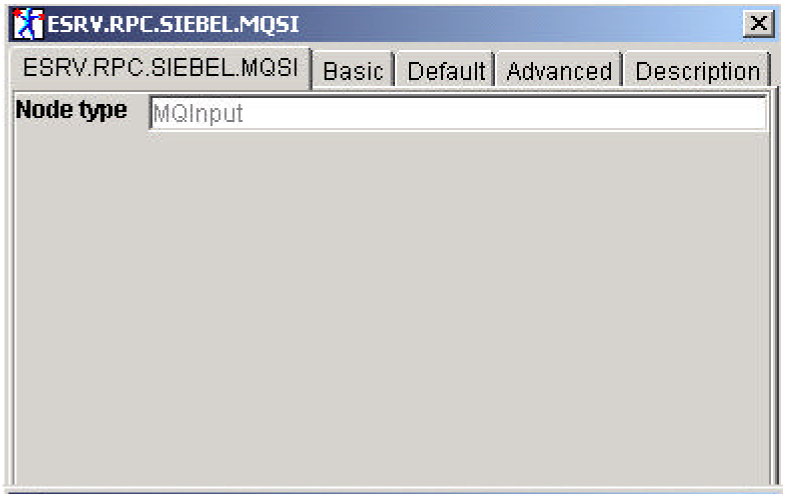
Usually customized using SQL

- Filter, Extract & Compute for message manipulation
- Database interaction
- Content-based filtering for subscriptions
- Based on SQL3 standard

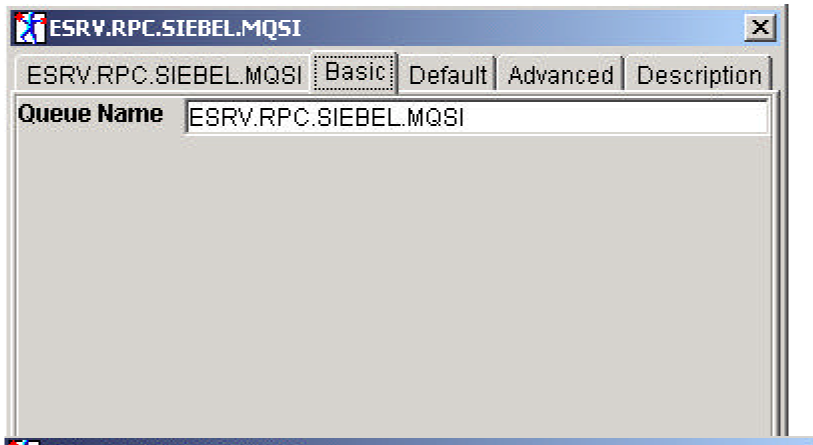
```
► SET OutputRoot.MQMD.UserIdentifier = 'jill';
► INSERT INTO output SELECT * FROM input WHERE
        filter_expression;
► DECLARE I INTEGER;
  SET I = 1;
  WHILE I < CARDINALITY(InputRoot.*[]) DO
        SET OutputRoot.*[I] = InputRoot.*[I];
        SET I=I+1;
  END WHILE;
► SELECT SUM
     (CAST(I.PRICE AS DECIMAL) * CAST(I.QUANTITY AS
  INTEGER) ) FROM Body.Invoice.Item[] AS I) > 100;
```

FEDERAL STUDENT AID
We Help Put America Through School

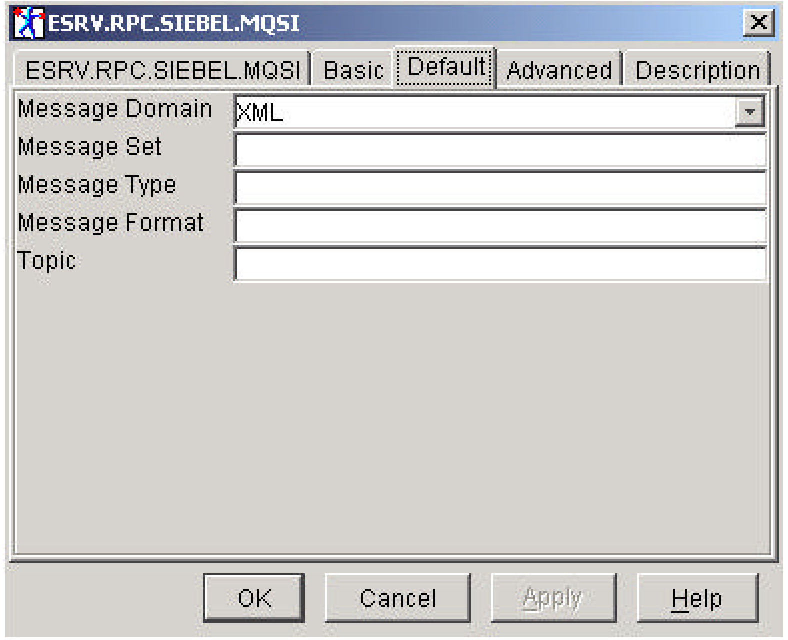
MQSI: Node Properties


ESRV.RPC.SIEBEL.MQSI

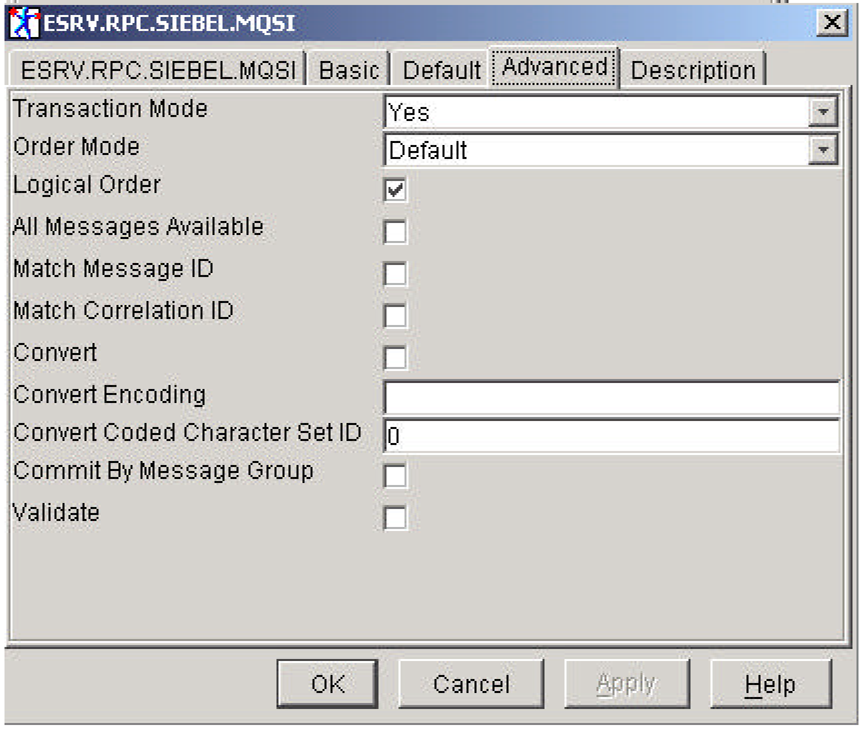
Node type	MQInput


ESRV.RPC.SIEBEL.MQSI

Queue Name	ESRV.RPC.SIEBEL.MQSI


ESRV.RPC.SIEBEL.MQSI

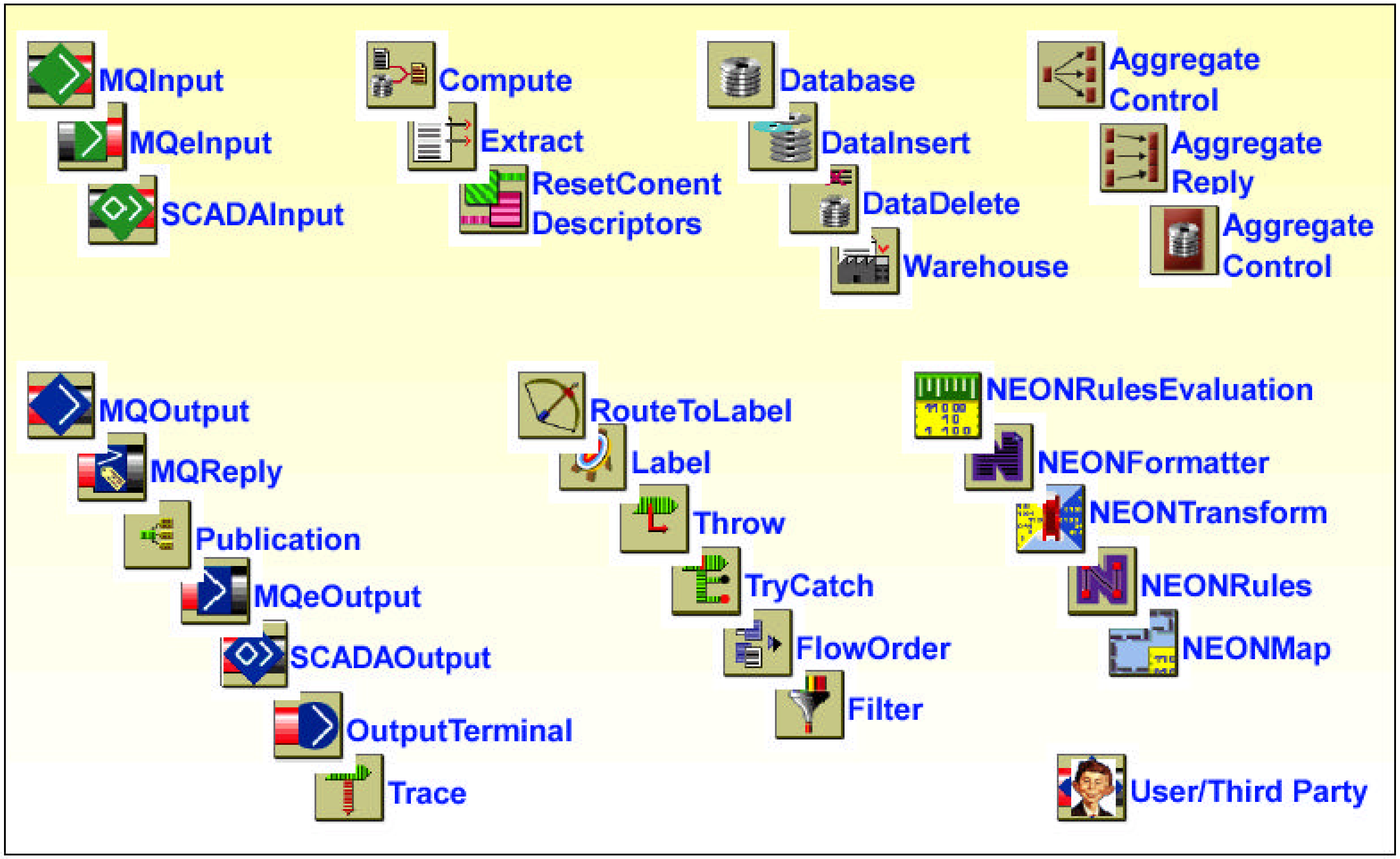
Message Domain	XML
Message Set	
Message Type	
Message Format	
Topic	


ESRV.RPC.SIEBEL.MQSI

Transaction Mode	Yes
Order Mode	Default
Logical Order	✓
All Messages Available	☐
Match Message ID	☐
Match Correlation ID	☐
Convert	☐
Convert Encoding	
Convert Coded Character Set ID	0
Commit By Message Group	☐
Validate	☐

FEDERAL STUDENT AID
We Help Put America Through School

MQSI: Primitive Nodes



MQSI: Nodes

Many "built-in" processing nodes

• Check	- Message format checking
• Compute	- Computation based on message fields and/or database lookup
• Database	- Database update and interaction
• Filter	- Filter based on message content and/or database lookup
• pass-through	- Simple pass through element
• Subscription	- Dynamic subscription management
• Throw	- Generate exception
• Trace	- Message trace logging element
• TryCatch	- Exception-trapping mechanism
• Warehouse	- Storage of message to warehouse including schema definition (specialization of Database)
• Neon rules/format	- Equivalent function to MQSeries Integrator V1.
• Neon transform	- Permits transform between message formats held in Neon message repository

Support for vendor "plug-in" nodes

Graphical construction tools

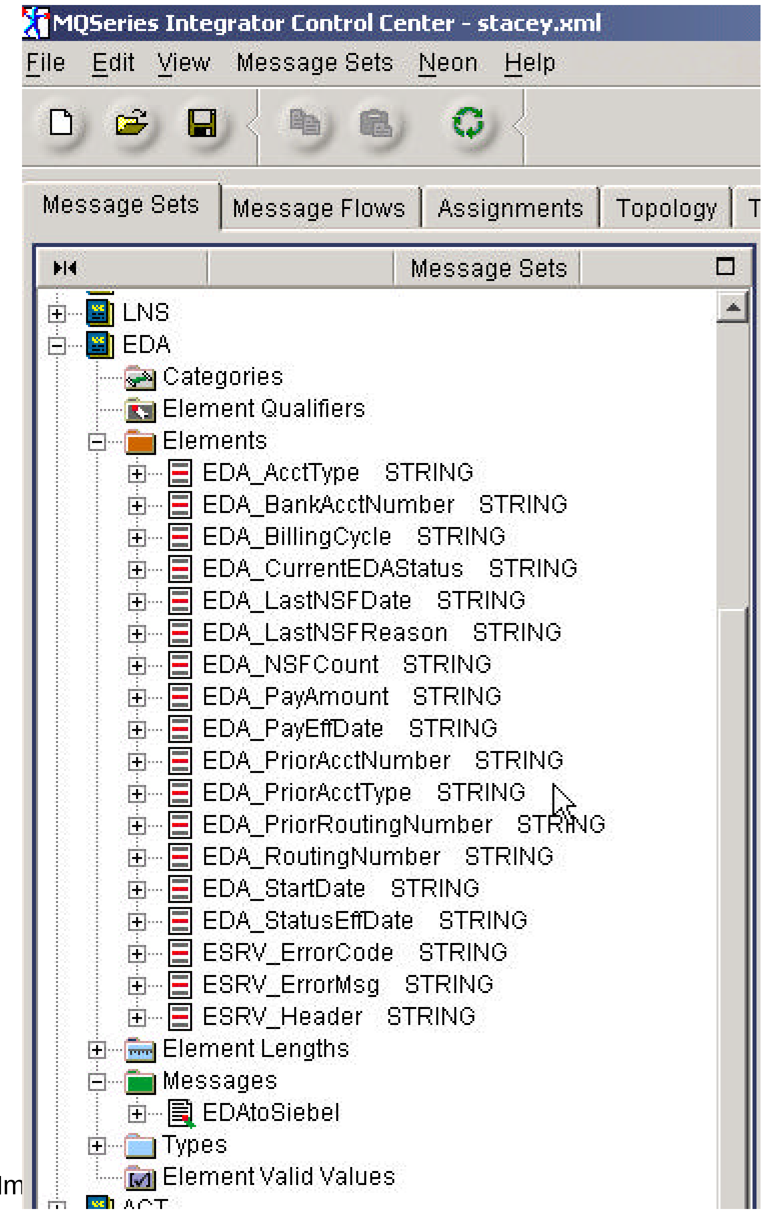
- Integrated development / deployment / management tools

Product ships with "pre-configured" message flows

- Basic publish/subscribe capability

FEDERAL STUDENT AID
We Help Put America Through School

MQSI: Message Set


MQSeries Integrator Control Center - stacey.xml

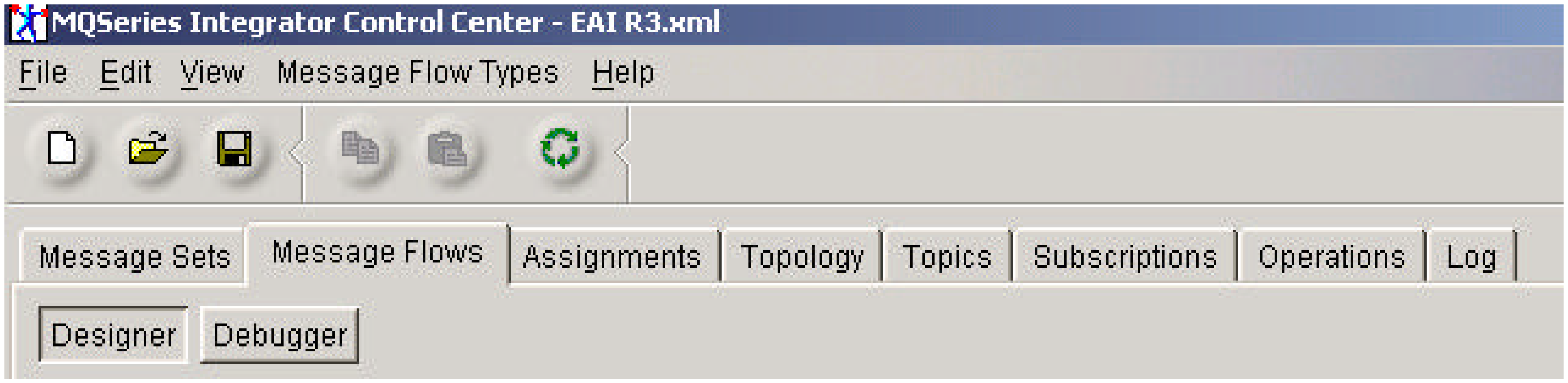
```
LNS
EDA
    Categories
    Element Qualifiers
    Elements
        EDA_AcctType   STRING
        EDA_BankAcctNumber   STRING
        EDA_BillingCycle   STRING
        EDA_CurrentEDAStatus   STRING
        EDA_LastNSFDate   STRING
        EDA_LastNSFReason   STRING
        EDA_NSFCount   STRING
        EDA_PayAmount   STRING
        EDA_PayEffDate   STRING
        EDA_PriorAcctNumber   STRING
        EDA_PriorAcctType   STRING
        EDA_PriorRoutingNumber   STRING
        EDA_RoutingNumber   STRING
        EDA_StartDate   STRING
        EDA_StatusEffDate   STRING
        ESRV_ErrorCode   STRING
        ESRV_ErrorMsg   STRING
        ESRV_Header   STRING
    Element Lengths
    Messages
        EDAtoSiebel
    Types
    Element Valid Values
ACT
```

Message Sets contain:

- Elements
- Types: Simple
  Compound
- Lengths
- Valid Values
- Messages

FEDERAL STUDENT AID
We Help Put America Through School

MQSI: Control Center



- Definition of Message Sets
- Definition of Message Flows
- Assignments
- Definition of the Topology
- Definition of topics
- Management of subscriptions
- Operations
- Log

Message Definition - Notes

The definition of message requires that the individual components of a message be defined and then included within other components. A message is made up of the following:

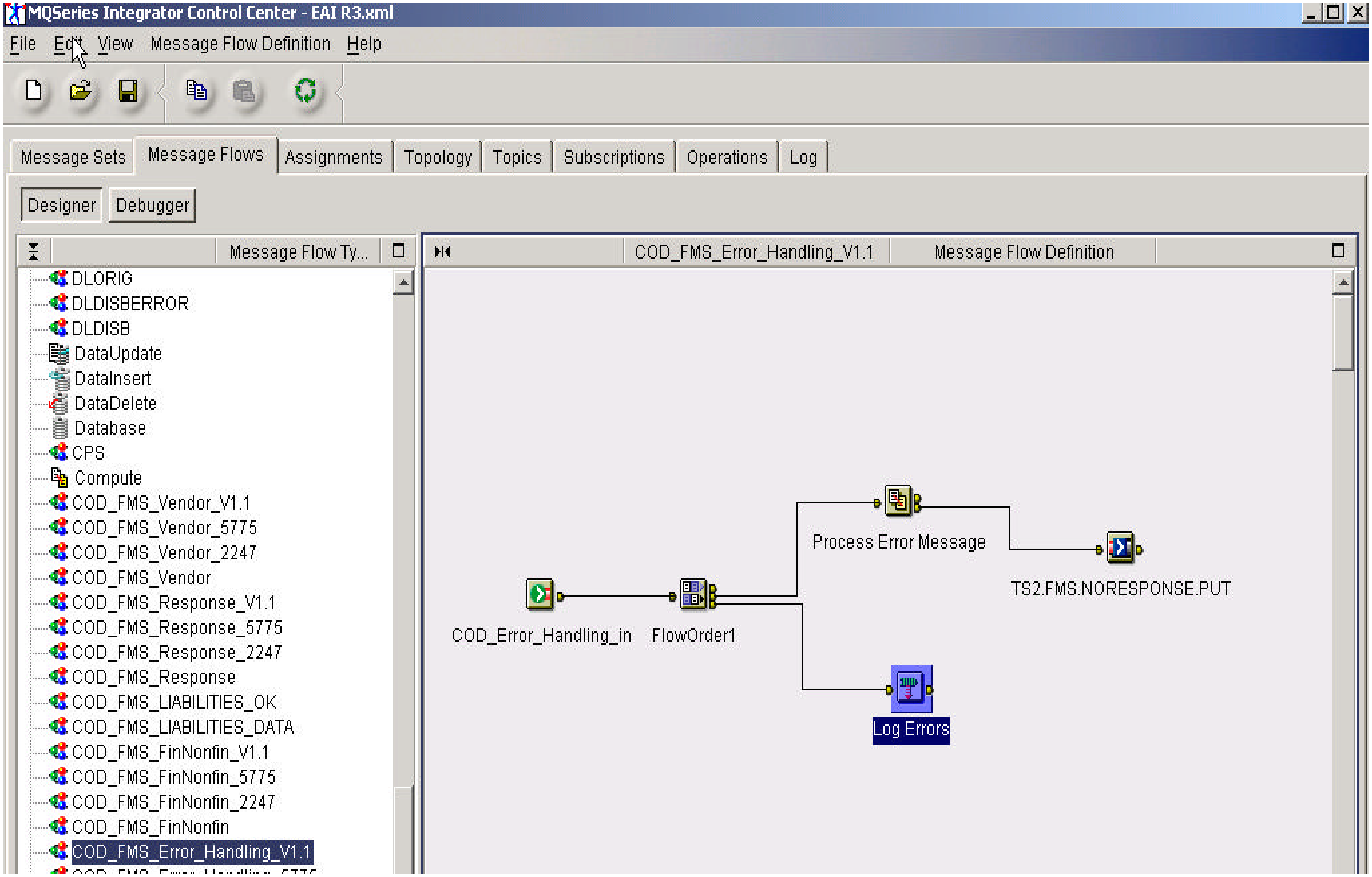
- Elements...also commonly referred to as fields.
  - Elements may have associated lengths
  - Elements may have associated valid values. These may be a range of values or several instances of different valid values to make up an enumeration list.
  - Each element has an associated type. This may be a simple type such as STRING or INTEGER or a compound type made up of other elements. Thus it is possible to construct arbitrarily nested elements (and, therefore, messages) composed of multiple compound types.
- Messages
  Messages are made up of a set of elements in a specific order.

Individual messages are then grouped together in a Message Set. The Message Set is the smallest 'unit' that is made available to a broker.

MQSI Message Repository Manager (MRM)

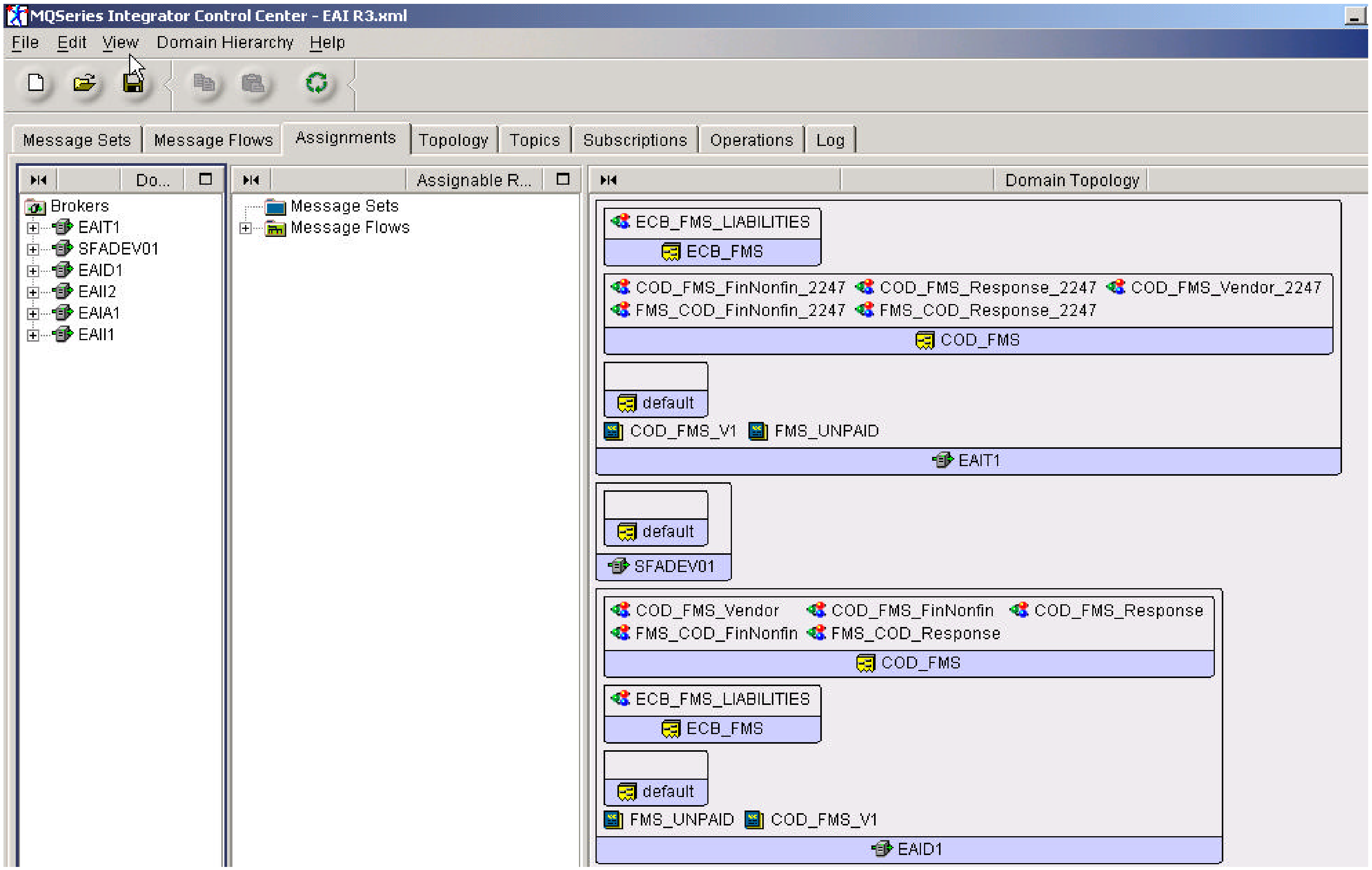
- Holds logical message format definitions
  - Definitions are grouped into message sets
  - Messages in a set share common structures and elements
- Contains mapping rules:
  - Mappings to XML and other message formats
- GUI to create and manage levels of message set
  - Project Management/Version control functions
- Extractors to produce
  - Dictionary file for Message Service Runtime
  - DTD
  - Message set documentation
- Importers for C/Cobol structures

MQSI:Msg Flow tab



FEDERAL STUDENT AID
We Help Put America Through School

MQSI:Assignments tab


MQSeries Integrator Control Center - EAI R3.xml
File Edit View Domain Hierarchy Help
Message Sets Message Flows Assignments Topology Topics Subscriptions Operations Log
Do...
Assignable R...
Domain Topology

Brokers
- EAIT1
- SFADEV01
- EAID1
- EAII2
- EAIA1
- EAII1

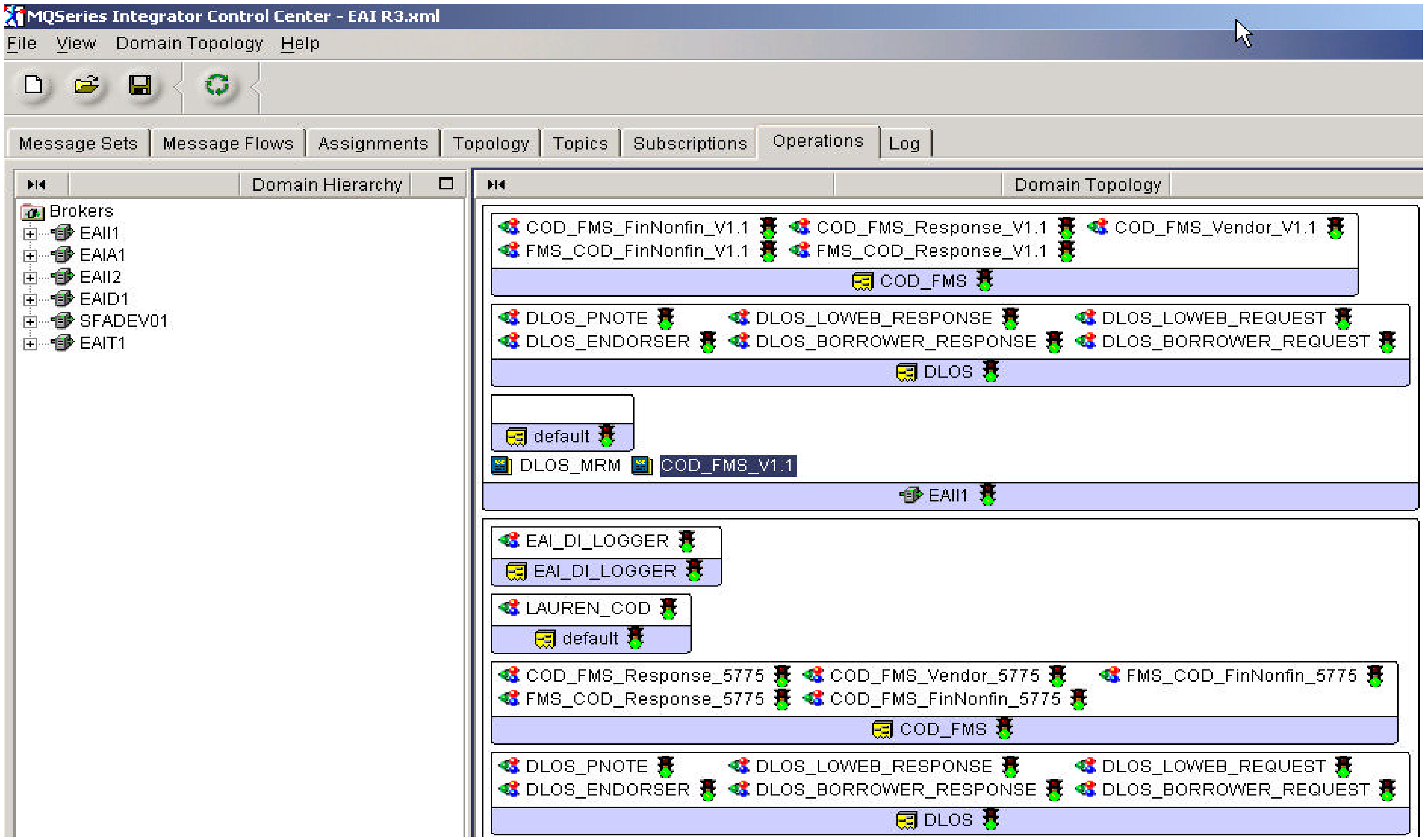
Message Sets
Message Flows

ECB_FMS_LIABILITIES
ECB_FMS
COD_FMS_FinNonfin_2247 COD_FMS_Response_2247 COD_FMS_Vendor_2247
FMS_COD_FinNonfin_2247 FMS_COD_Response_2247
COD_FMS
default
COD_FMS_V1 FMS_UNPAID
EAIT1

default
SFADEV01

COD_FMS_Vendor COD_FMS_FinNonfin COD_FMS_Response
FMS_COD_FinNonfin FMS_COD_Response
COD_FMS
ECB_FMS_LIABILITIES
ECB_FMS
default
FMS_UNPAID COD_FMS_V1
EAID1

MQSI: Operations

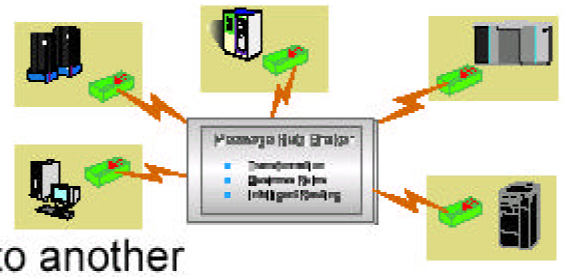

MQSeries Integrator Control Center - EAI R3.xml

Brokers
- EAII1
- EAIA1
- EAII2
- EAID1
- SFADEV01
- EAIT1

EAII1
- COD_FMS: COD_FMS_FinNonfin_V1.1, COD_FMS_Response_V1.1, COD_FMS_Vendor_V1.1, FMS_COD_FinNonfin_V1.1, FMS_COD_Response_V1.1
- DLOS: DLOS_PNOTE, DLOS_LOWEB_RESPONSE, DLOS_LOWEB_REQUEST, DLOS_ENDORSER, DLOS_BORROWER_RESPONSE, DLOS_BORROWER_REQUEST
- default
- DLOS_MRM
- COD_FMS_V1.1

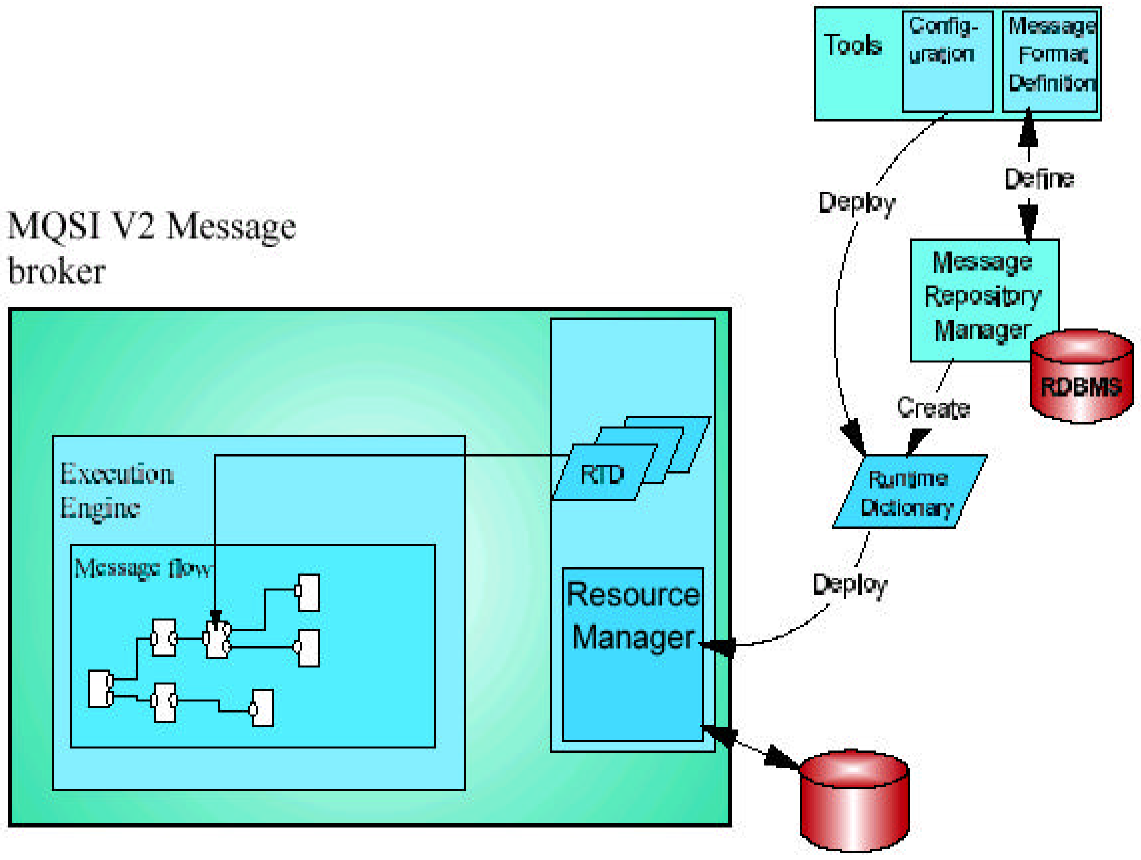
- EAI_DI_LOGGER: EAI_DI_LOGGER
- default: LAUREN_COD
- COD_FMS: COD_FMS_Response_5775, COD_FMS_Vendor_5775, FMS_COD_FinNonfin_5775, FMS_COD_Response_5775, COD_FMS_FinNonfin_5775
- DLOS: DLOS_PNOTE, DLOS_LOWEB_RESPONSE, DLOS_LOWEB_REQUEST, DLOS_ENDORSER, DLOS_BORROWER_RESPONSE, DLOS_BORROWER_REQUEST

XML and MQSeries Integrator



- MQSI V2 provides XML message broking
  - Change contents of an XML message
  - Transform from one XML message format to another
  - Transform between XML and non-XML formats
  - Create new XML messages
  - Filter and Route XML messsage based on their content
  - Allow Pub/Sub Subscriptions against XML content
  - Augment XML messages with data from a database
  - Update database with data from an XML message
- MQSI V2 Message Repository support
  - Messages can be modelled and checked at runtime
- MQSI V2 uses XML internally
  - Control Center/Runtime communication

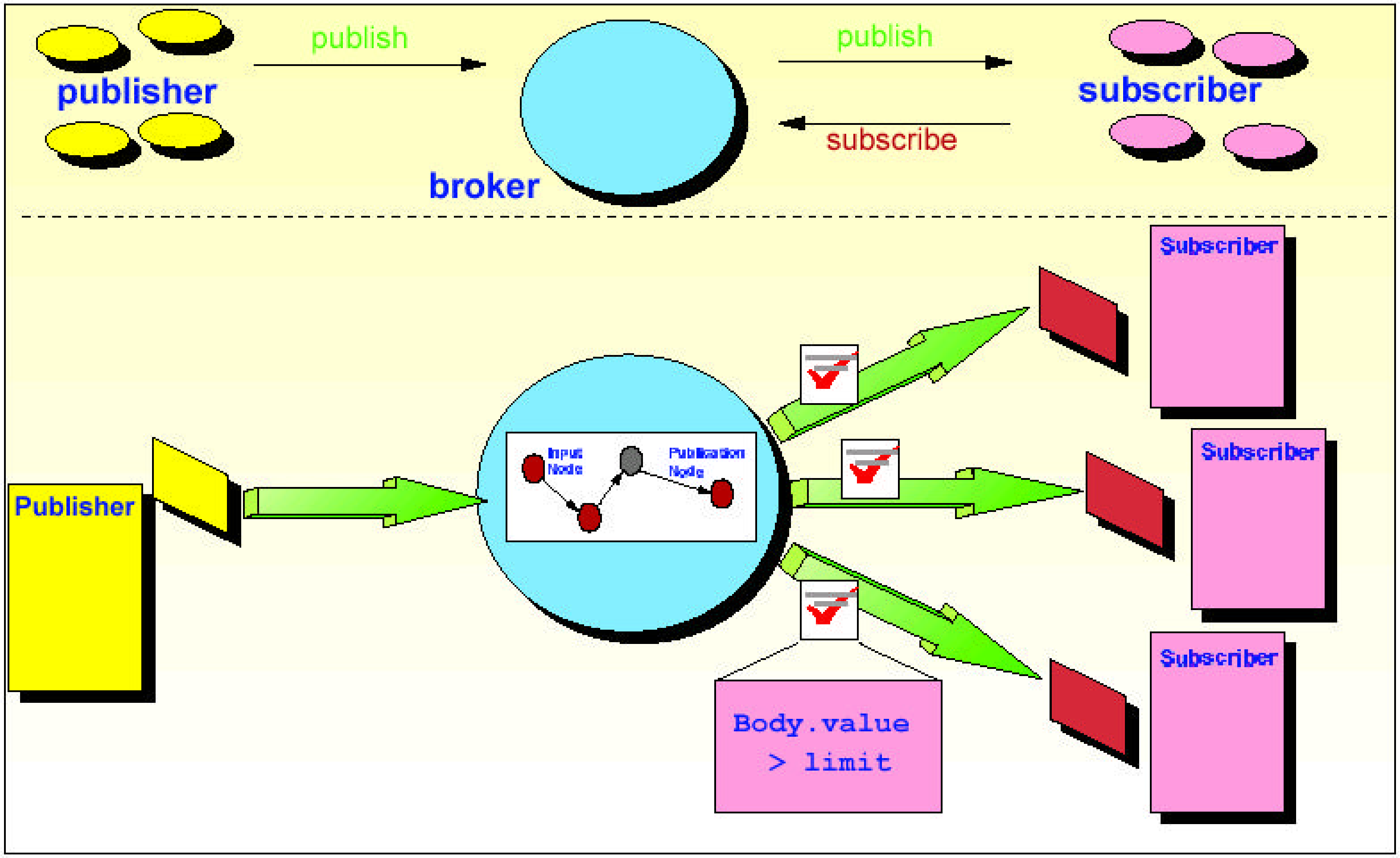
Message Format Service - Deployment



FEDERAL STUDENT AID
We Help Put America Through School

MQSI: Publish/Subscribe

Publish/Subscribe - Publications, Subscriptions and Filters



MQSI: Pub/Sub

MQSI supports 2 application models:

Point-to-point and publish/subscribe

Point-to-point applications exchange info with known partners. Each app is aware of the applications to which it is logically connected.

Publish/Subscribe apps are not tied to particular partners; they use msgs that have more flexible delivery requirements in terms of their origins and destinations. Msgs are published about a particular topic, rather than to a particular recipient. Msgs are available at any time, for any interested receiver without it being aware of the sender. This application architecture is more dynamic and anonymous than point-to-point messaging.

FEDERAL STUDENT AID
We Help Put America Through School

Data Integrator

Provides the means to exchange information between dissimilar

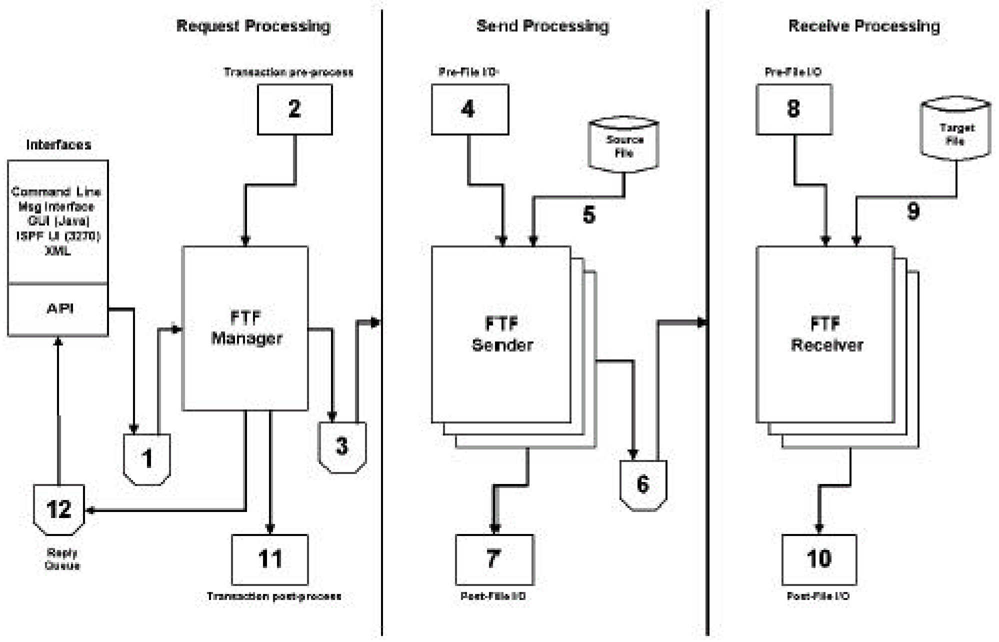
Networks

Operating systems

Databases

Applications

Data Integrator: Flow

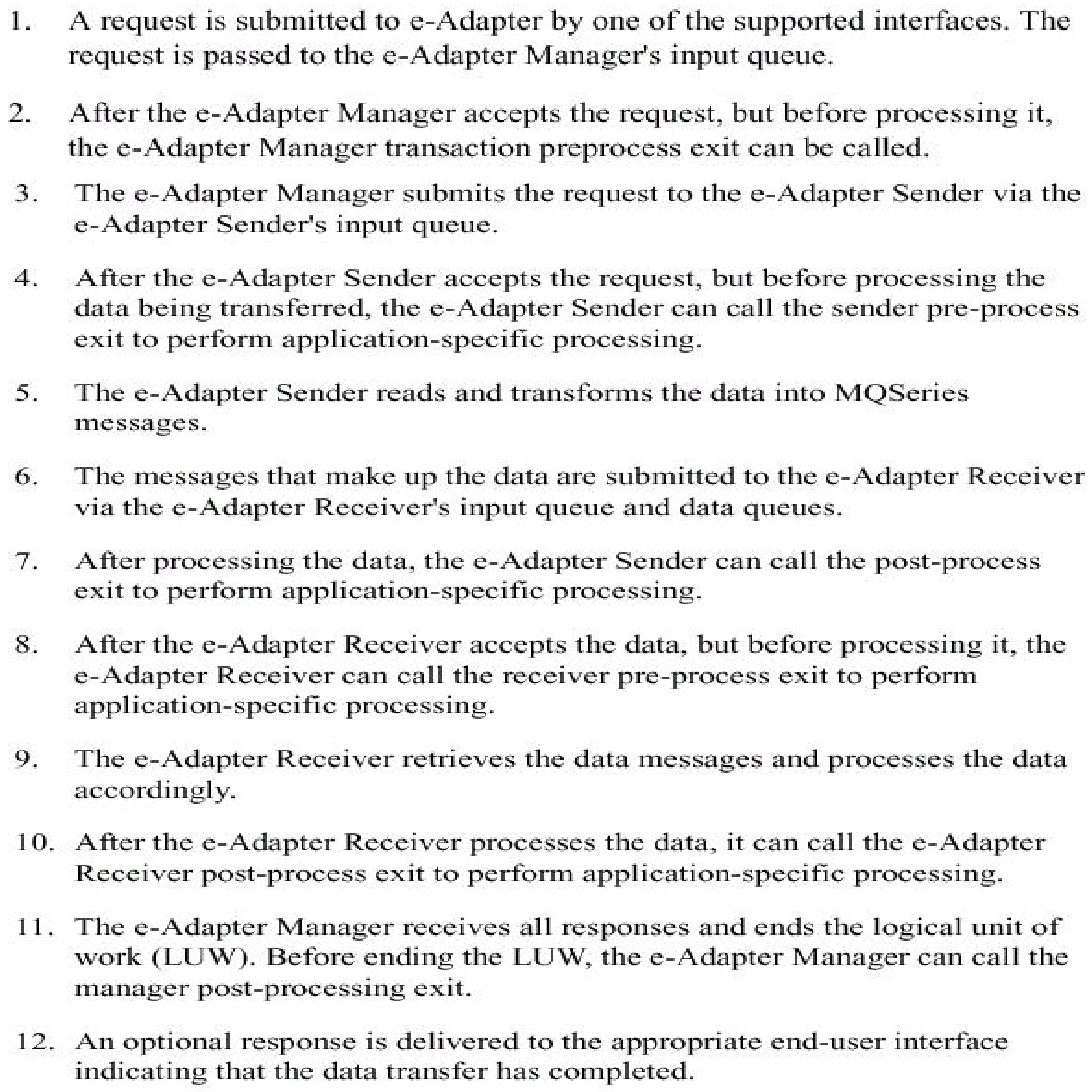


The following major components compose e-Adapter:

- e-Adapter Interfaces
- e-Adapter Manager
- e-Adapter Sender
- e-Adapter Receiver
- e-Adapter Status

FEDERAL STUDENT AID
We Help Put America Through School

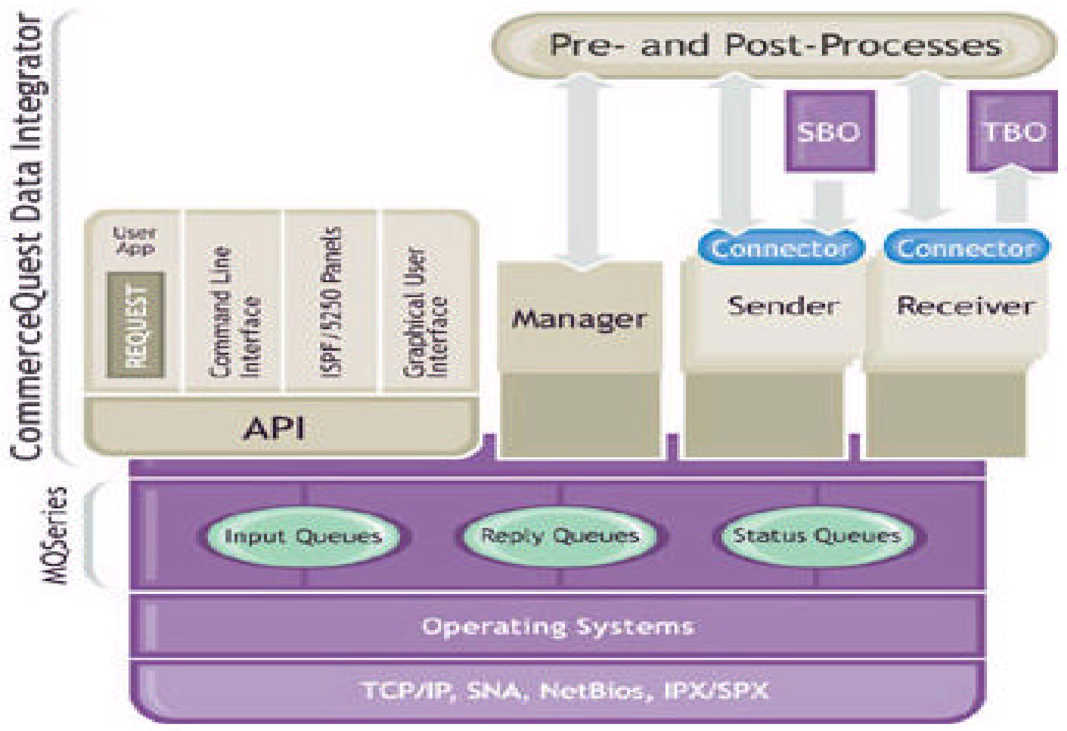
DI: Flow



1. A request is submitted to e-Adapter by one of the supported interfaces. The request is passed to the e-Adapter Manager's input queue.
2. After the e-Adapter Manager accepts the request, but before processing it, the e-Adapter Manager transaction preprocess exit can be called.
3. The e-Adapter Manager submits the request to the e-Adapter Sender via the e-Adapter Sender's input queue.
4. After the e-Adapter Sender accepts the request, but before processing the data being transferred, the e-Adapter Sender can call the sender pre-process exit to perform application-specific processing.
5. The e-Adapter Sender reads and transforms the data into MQSeries messages.
6. The messages that make up the data are submitted to the e-Adapter Receiver via the e-Adapter Receiver's input queue and data queues.
7. After processing the data, the e-Adapter Sender can call the post-process exit to perform application-specific processing.
8. After the e-Adapter Receiver accepts the data, but before processing it, the e-Adapter Receiver can call the receiver pre-process exit to perform application-specific processing.
9. The e-Adapter Receiver retrieves the data messages and processes the data accordingly.
10. After the e-Adapter Receiver processes the data, it can call the e-Adapter Receiver post-process exit to perform application-specific processing.
11. The e-Adapter Manager receives all responses and ends the logical unit of work (LUW). Before ending the LUW, the e-Adapter Manager can call the manager post-processing exit.
12. An optional response is delivered to the appropriate end-user interface indicating that the data transfer has completed.

FEDERAL STUDENT AID
We Help Put America Through School

DI: Components



FEDERAL STUDENT AID
We Help Put America Through School

e-Adapter Manager

- ❖ Manages status of all transfers
  - reads its input queue
  - creates log entries
  - submits stat msgs to queues
- ❖ Starts and stops all transfer units of work
- ❖ Correlates all operational replies and reports final status of the transaction (not to be confused with status msgs)
  - Request completed successfully
  - Request failed
  - Request expired
  - Request canceled

Sender

- ❖ Transforms the data into MQSeries messages
  - reads its input queue
  - creates log entries
  - submits status messages
- ❖ The sender is always where the source data resides
- ❖ Updates the e-Adapter Manager with operational replies

DI: e-Adapter Rcvr

Receiver

- ❖ Receives incoming data from MQSeries.
  - reads its input queue
  - creates log entries
  - submits status messages
- ❖ The receiver is always the destination for the data
- ❖ Updates the e-Adapter Manager with operational replies

Status

- Status messages are received by each of the e-Adapter components and are not necessary for internal e-Adapter processing
- Status messages provide a reporting system that report on current and past status of data-transfer requests
- Status messages are MQSeries messages destined for the queue or list of queues defined in the e-Adapter config file

DI: Exits

- ❖E-Adapter provides several exit points at strategic locations during the data-transfer request.
- ❖ Must be developed in C
- ❖ User exits are invoked synchronously by the e-Adapter components.

POOLS

- ❖ e-Adapter Pools use logical queues made up of muliple physical queues
- ❖ Pools allow
  - Increased throughput
  - Segregation of data-transfer traffic
  - Overcome MQSeries queue capacity limitations

FEDERAL STUDENT AID
We Help Put America Through School

DI: Available Exits

Exit #	Executing Node	E-Adapter Component	Description
Exit 3	oqm	Mgr	Mgr preprocess exit. If specified, this is the 1st exit executed in the data transfer
Exit 4	oqm	Mgr	Mgr post-process. If specified, this is the last exit executed in the data tranfer
Exit 5	sqm	Sdr	Sdr pre-process exit. Invoked before the e-Adptr Sdr reads the source file
Exit 6	sqm	Sdr	Sdr post-process exit. Invoked after the source file’s contents have been read
Exit 7	dqm	Rcv	Rcvr pre-process exit. Invoked before the rcvr begins to write the target file
Exit 8	dqm	Rcv	Rcvr post-process exit. Inovked after the target file has been processed
Exit 9	sqm	Sdr	Invokes sdr connector exit
Exit 10	dqm	Rcv	Invokes rcvr connector exit

DI and FSA

- Current uses and interfaces

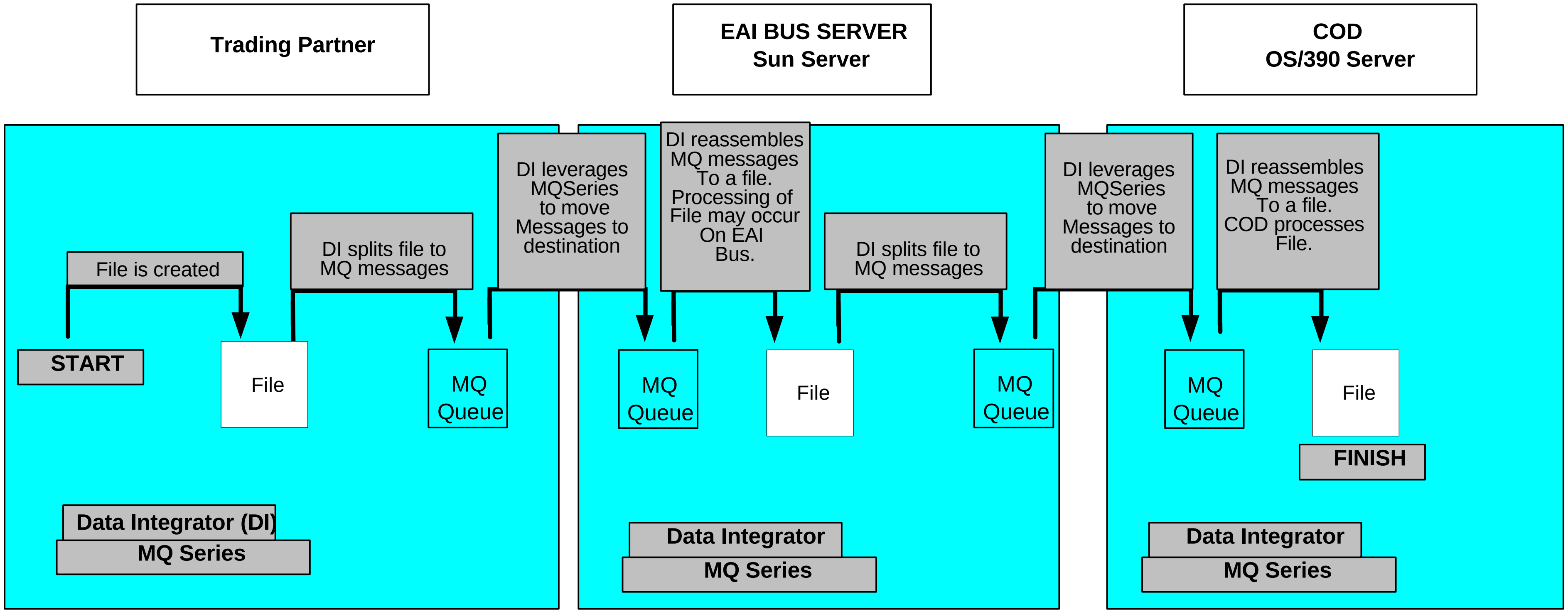
DI: Subsystem

❖ The e-Adapter subsystem provides the following services

- Moves and accepts files among supported services
- Provides data compression if you require it
- Performs binary and ascii transfers
- Transfers files regardless of size, format, or destination
- Allows individual status tracking for any phase of the file transfer at any node across the enterprise

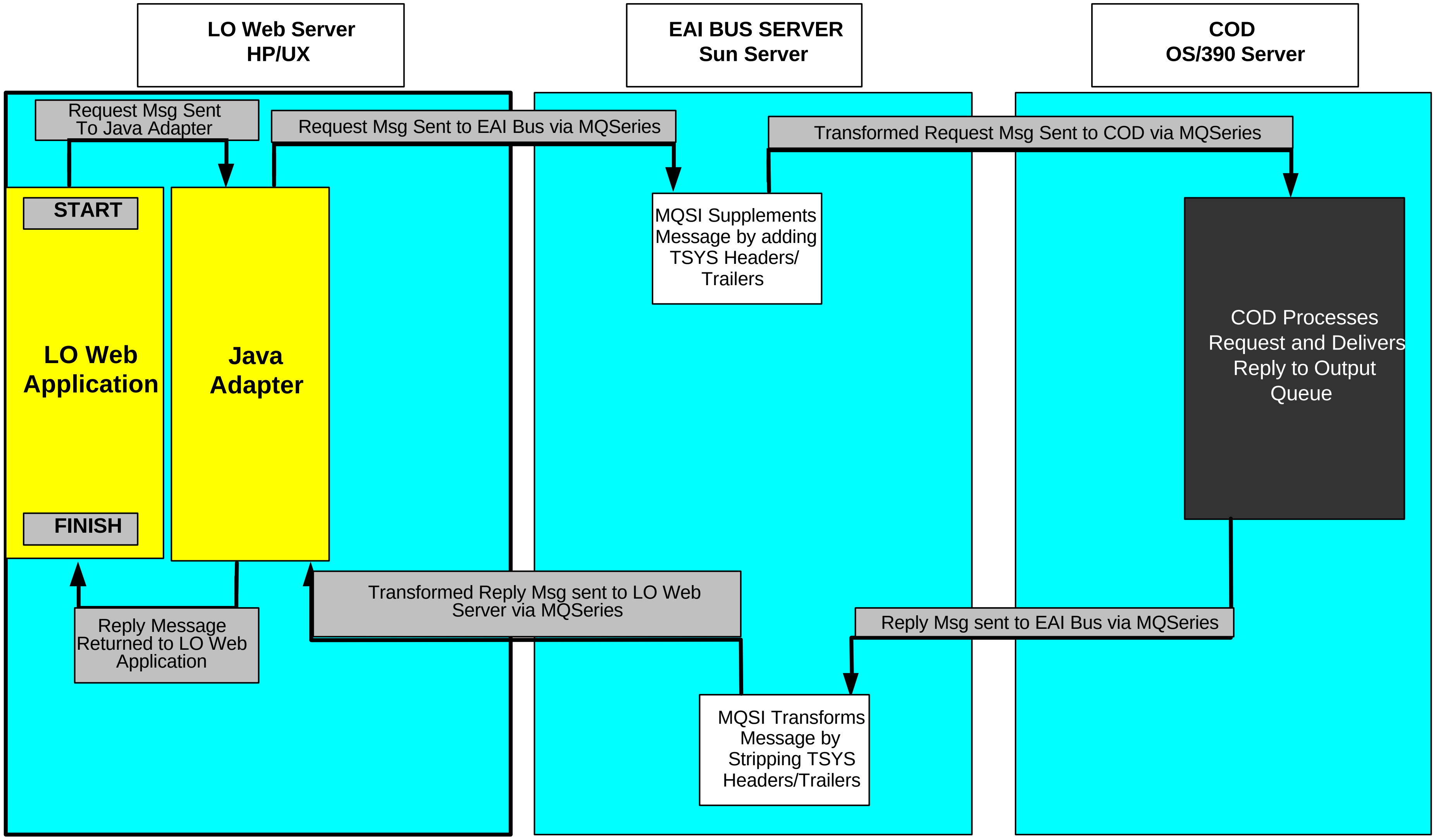
FEDERAL STUDENT AID
We Help Put America Through School

Files are transferred from trading partners to COD through the services offered by the EAI architecture.


Trading Partner
EAI BUS SERVER
Sun Server
COD
OS/390 Server
File is created
START
File
DI splits file to MQ messages
MQ Queue
DI leverages MQSeries to move Messages to destination
DI reassembles MQ messages To a file. Processing of File may occur On EAI Bus.
MQ Queue
File
DI splits file to MQ messages
MQ Queue
DI leverages MQSeries to move Messages to destination
DI reassembles MQ messages To a file. COD processes File.
MQ Queue
File
FINISH
Data Integrator (DI)
MQ Series
Data Integrator
MQ Series
Data Integrator
MQ Series

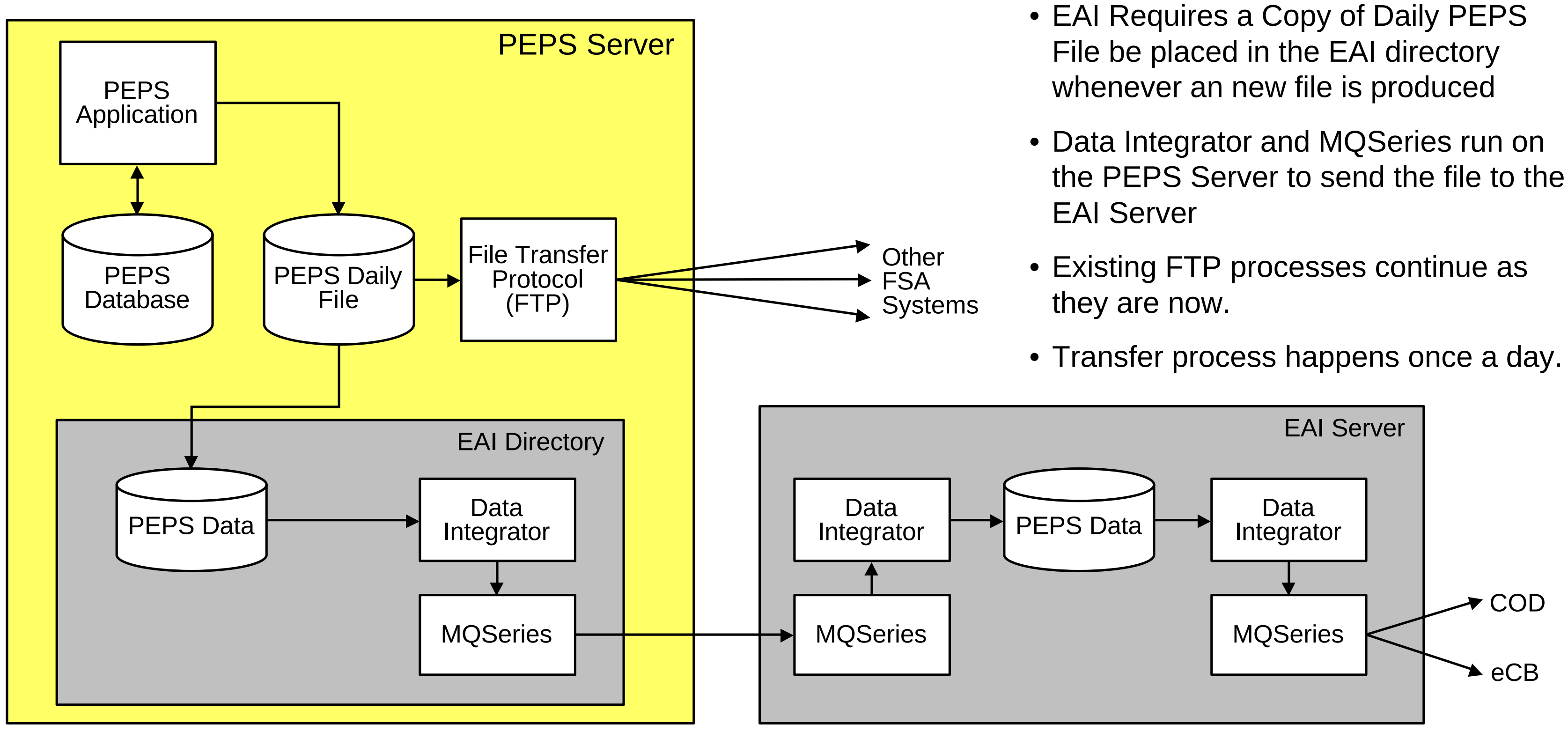
FEDERAL STUDENT AID
We Help Put America Through School

Individual transactions are transported and transformed, if necessary, using the services provided by the EAI architecture.



LO Web Server HP/UX
EAI BUS SERVER Sun Server
COD OS/390 Server
Request Msg Sent To Java Adapter
Request Msg Sent to EAI Bus via MQSeries
Transformed Request Msg Sent to COD via MQSeries
START
LO Web Application
FINISH
Java Adapter
MQSI Supplements Message by adding TSYS Headers/ Trailers
COD Processes Request and Delivers Reply to Output Queue
Transformed Reply Msg sent to LO Web Server via MQSeries
Reply Msg sent to EAI Bus via MQSeries
Reply Message Returned to LO Web Application
MQSI Transforms Message by Stripping TSYS Headers/Trailers

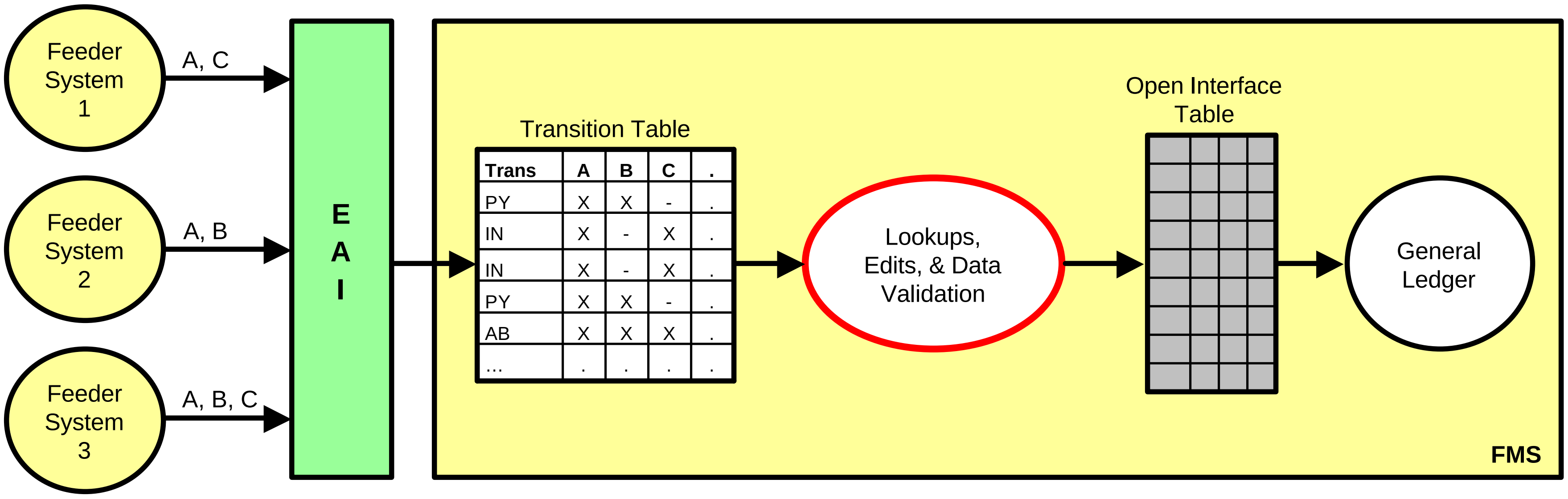
The PEPS EAI Interface Minimizes the Impact on PEPS Resources



- EAI Requires a Directory and Workspace on PEPS Server
- EAI Requires a Copy of Daily PEPS File be placed in the EAI directory whenever an new file is produced
- Data Integrator and MQSeries run on the PEPS Server to send the file to the EAI Server
- Existing FTP processes continue as they are now.
- Transfer process happens once a day.

FEDERAL STUDENT AID
We Help Put America Through School

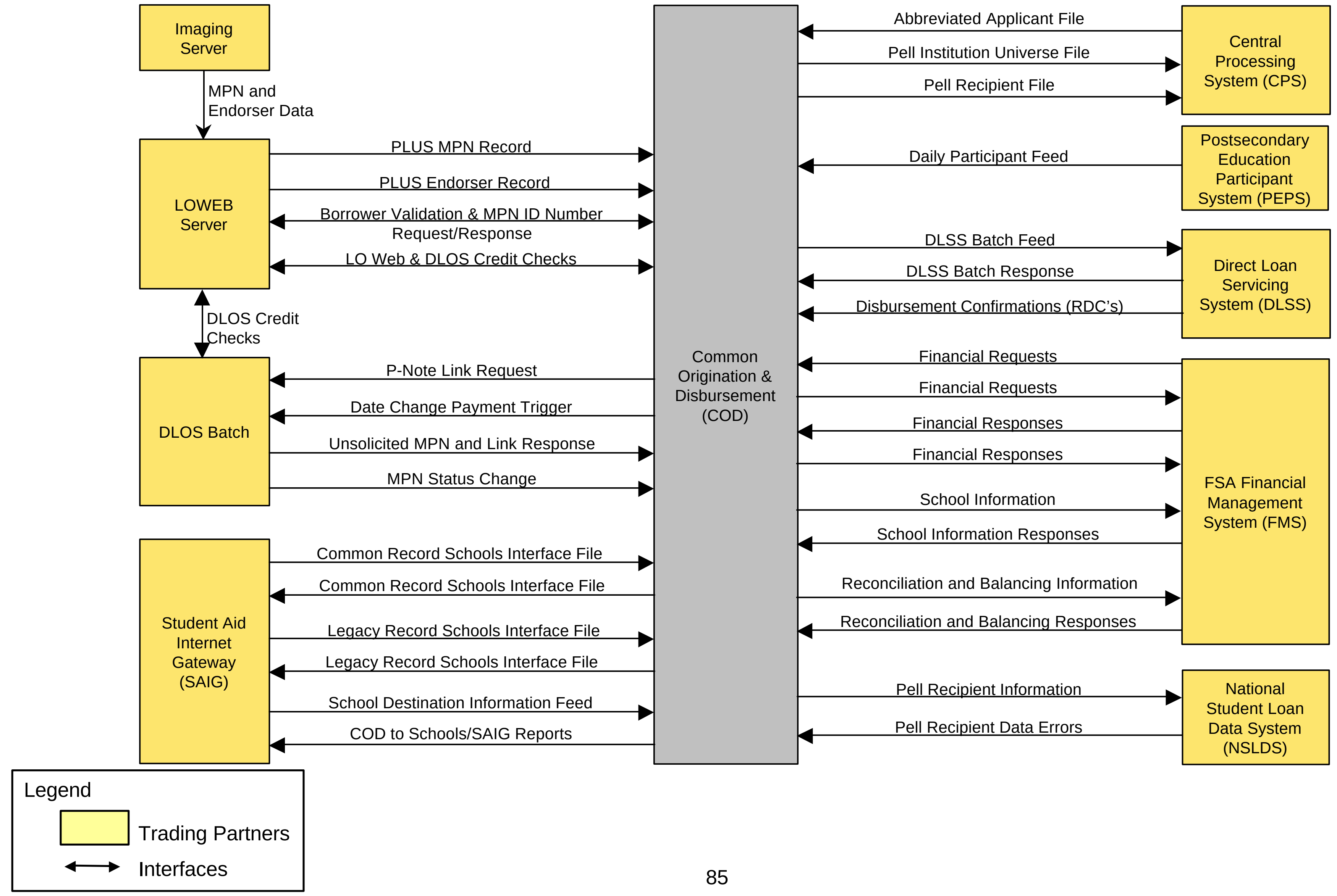
The FMS - EAI Interface Approach Utilizes a Transition Table to Map Feeder System Data Elements Into FMS



Trans	A	B	C	.
PY	X	X	-	.
IN	X	-	X	.
IN	X	-	X	.
PY	X	X	-	.
AB	X	X	X	.
...	.	.	.	.

Feeder Systems	EAI	Transition Table	Data Loading	OI Table	FMS Module
• Provide data according to own format • Different systems may provide different data elements	• Routes data to appropriate Transition Table and Column • Converts data formats as necessary • Validates basic data formats	• Contains columns for all possible data elements • Source system is implied by the data	• Performs feeder specific processing • Performs table lookups • Performs data validation and edits	• Presents input data for Module processing • One Open Interface Table for each FMS Module	• One FMS Module each for General Ledger, Accounts Payable, Accounts Receivable

COD is composed of multiple interfaces with FSA's trading partners.



Imaging Server
MPN and Endorser Data
LOWEB Server
PLUS MPN Record
PLUS Endorser Record
Borrower Validation & MPN ID Number Request/Response
LO Web & DLOS Credit Checks
DLOS Credit Checks
DLOS Batch
P-Note Link Request
Date Change Payment Trigger
Unsolicited MPN and Link Response
MPN Status Change
Student Aid Internet Gateway (SAIG)
Common Record Schools Interface File
Common Record Schools Interface File
Legacy Record Schools Interface File
Legacy Record Schools Interface File
School Destination Information Feed
COD to Schools/SAIG Reports
Common Origination & Disbursement (COD)
Abbreviated Applicant File
Pell Institution Universe File
Pell Recipient File
Central Processing System (CPS)
Daily Participant Feed
Postsecondary Education Participant System (PEPS)
DLSS Batch Feed
DLSS Batch Response
Disbursement Confirmations (RDC's)
Direct Loan Servicing System (DLSS)
Financial Requests
Financial Requests
Financial Responses
Financial Responses
School Information
School Information Responses
Reconciliation and Balancing Information
Reconciliation and Balancing Responses
FSA Financial Management System (FMS)
Pell Recipient Information
Pell Recipient Data Errors
National Student Loan Data System (NSLDS)
Legend
Trading Partners
Interfaces